UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES "ARAGON"

"DISEÑO DE SISTEMAS DIGITALES CON LOGICA PROGRAMABLE UTILIZANDO LA TARJETA SPARTAN 3E"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A:

JOSE ANGEL CHAVEZ GARCIA

ASESOR:

ING. MARTIN HERNÁNDEZ HERNANDEZ

MÉXICO MARZO DEL 2008

“Agradezco a la UNAM
Agradezco a mi familia,
Doy gracias a dios”

Mis pilares

DISEÑO DE SISTEMAS DIGITALES CON LOGICA PROGRAMABLE UTILIZANDO LA TARJETA SPARTAN 3E.

CONTENIDO Pag.

CAPITULO 5: EJEMPLOS

INTRODUCCIÓN

La necesidad de construir circuitos digitales cada vez más complejos es patente día a día. Ya en el siglo XXI somos capaces de construir microprocesadores de muy altas prestaciones que están compuestos por millones de unidades funcionales (transistores) que realizan tareas de gran responsabilidad en la sociedad. Por ejemplo, un sistema de control de transacciones económicas de una bolsa de valores ha de ser un sistema informático extraordinariamente rápido y robusto, ya que un fallo en la transferencia de información acarrearía un sinfín de problemas con los inversores. Otro ejemplo, la electrónica de control de un avión supersónico tiene igualmente una responsabilidad extrema, tanto en aportar la información necesaria al piloto para determinar su rumbo como para asistirle en sus tareas de pilotaje y combate.

En la práctica, el 100% de la electrónica de control y supervisión de los sistemas, elaboración de datos y transferencia de los mismos se realiza mediante circuitos integrados digitales, constituidos por una gran cantidad de transistores: son los llamados circuitos integrados de muy alta escala de integración, o VLSI. Si en los años cincuenta y sesenta, en los albores de la electrónica integrada los circuitos eran esencialmente analógicos, en los que el número de elementos constituyentes de los circuitos no pasaba de la centena, en la actualidad el hombre dispone de tecnologías de integración capaces de producir circuitos integrados con millones de transistores a un coste no muy elevado, al alcance de una PYME.

A mediados de los años sesenta Gordon E. Moore ya vaticinaba un desarrollo de la tecnología planar en el que cada año la escala de integración se doblaría, y de la misma manera aumentaría la capacidad de integrar funciones más complejas y la velocidad de procesamiento de esas funciones. Las predicciones de Moore se han cumplido con gran exactitud durante los siguientes 30 años, y que la tendencia continuará durante los próximos 20. En el año 2012 Intel espera integrar 1000 millones de transistores funcionando a 10GHz.

Si bien construir estos circuitos parece una cuestión superada, diseñarlos supone un serio problema. Los primeros circuitos integrados eran diseñados a partir del trazado directo de las máscaras sobre un editor de *layout*, y prácticamente no eran comprobados antes de fabricarse. Se confiaba en la pericia del diseñador a la hora de elaborar los dibujos. Con la aparición de los primeros ordenadores de entonces gran potencia, llamados *estaciones de trabajo* (concepto que hoy en día no difiere sustancialmente del de ordenador personal) se incorporaron complejos programas de resolución de ecuaciones diferenciales que permitían, alimentados con un *modelo matemático* del circuito, verificar su funcionalidad antes de la fabricación. Este esquema funcionaba con circuitos digitales, y aún funciona, con circuitos analógicos, con escaso número de elementos muy bien dimensionados para determinar fielmente su comportamiento. La microelectrónica digital continúa por otro camino su desarrollo con herramientas específicas, como son los simuladores digitales o los generadores automáticos de layout, que resuelven el problema de la construcción del circuito y la verificación del mismo.

Nace así la ingeniería de computación o CAE en las que se delegan en herramientas software las tareas de manejo de grandes cantidades de información, bases de datos que, de forma óptima contienen la información acerca de la funcionalidad del circuito, de su geometría y de su conexiones así como de su comportamiento eléctrico.

Si bien, por un lado la electrónica digital supone una simplificación funcional de un comportamiento analógico, el tamaño de los circuitos digitales es una complicación que requiere una visión del problema muy diferente.

Una vez comentada la incidencia de la tecnología nos centramos en la especificación de la funcionalidad en sí, lo que se ha dado en llamar la interrelación hombre-máquina. Tradicionalmente se han utilizado los editores de esquemas, como parte del flujo de CAE o secuencia de programas que se han de utilizar para construir nuestro circuito integrado. Estos editores son un magnífico mecanismo de proyección de un diagrama de símbolos (tradicionalmente es el lenguaje utilizado por la electrónica) para expresar la funcionalidad deseada. Estos editores tienen un nivel de desarrollo espectacular. Son capaces de dar una visión muy precisa y completa del diseño rápidamente. A esto ha contribuido en gran manera el auge de los entornos gráficos de los sistemas operativos al estilo del conocido Windows, que a finales de lo ochenta tenía ya unos predecesores de gran potencia y prestaciones.

Sin embargo la complejidad de los circuitos digitales aumentaba y las prestaciones de los editores de esquemas no eran suficientes para responder a una capacidad de diseño tan elevada. Editar un esquema requiere un esfuerzo de desarrollo muy alto. Para una determinada función:

- La función ha de presentarse sin errores en ninguna conexión y función lógica. Se precisa una verificación, por simple que sea el módulo.

- Las señales han de tener asociadas un nombre significativo que permita su posterior identificación.

- Se construye a partir de unos elementos funcionales contenidos en una librería que proporciona un fabricante y por tanto ligada al mismo.

- La edición se hace con una interacción ratón, teclado, paleta de dibujo... etc. que ralentiza mucho el proceso de inserción del esquema.

- Como se puede observar la técnica de los esquemas es suficiente, pero requiere para cada unidad un gran esfuerzo y tiempo. ¿Que hacer ante este panorama?

A principios de los años 90 "*Cadence Design Systems"*, líder mundial en sistemas de CAE para microelectrónica, propone el Verilog, un lenguaje alfanumérico para describir los circuitos de forma sencilla y precisa: es el primer lenguaje de descripción de hardware en sentido amplio como veremos en epígrafes posteriores. Otros fabricantes de hardware habían propuesto un lenguaje más centrado en la resolución de un problema concreto: generación de una función para un dispositivo programable, resolución del circuito de una máquina de estados finitos a partir de su descripción de la evolución de los estados, etc. Nacen los conceptos de descripción de alto nivel y de síntesis lógica.

En el año 1982 el Departamento de Defensa de los Estados Unidos promueve un proyecto para desarrollar un lenguaje de descripción (conocido como MIL-STD-454L) de hardware que:

- Describiera los circuitos digitales de forma amplia: Funcionalidad, tecnología y conexionado

- Permitiera describir y verificar los circuitos a todos los niveles: funcional, arquitectural y tecnológico.

- Describiera la tecnología misma, para poder diseñar circuitos que sean independientes de la propia tecnología o bien durante la puesta a punto del proceso de fabricación.

- Describiera modelos del entorno en el que se va a insertar el circuito de forma que hubiese unas posibilidades de verificación más amplias del propio circuito.

JUSTIFICACION

La tendencia generalizada del uso de sistemas electrónicos en todos los campos de la técnica, han llevado a su evolución a un punto tal, en el que el desarrollo de sistemas digitales de propósitos específicos requieren de herramientas de implementación mas avanzadas. En este sentido, el acercamiento a los lenguajes de descripción de circuitos se hace necesario, así como también el manejo de tecnologías de ultima generación.

En este sentido, una de las tecnologías que están emergiendo son unos dispositivos llamados FPGA's ("Fiel Array Gate's Programmable") – Arreglo de compuertas programables en campo, dichos dispositivos comienzan a inundar el mercado de los microchips debido en parte a que representan ventajas en cuanto a la complejidad del sistema que puede implementarse y a que es reprogramable. Existen diversos fabricantes de FPGA's entre los que destacan: Xilinx y Actel. Se eligió desarrollar el presente trabajo con una tarjeta de desarrollo de Xilinx: La Spartan 3E Starter Kit, debido principalmente al relativo bajo costo de la misma, así como el grado de comercialización que tiene a nivel local.

El desarrollo práctico en el proceso de diseño por medio de lenguajes de descripción de circuitos, resulta en un conocimiento más concienzudo, y obtener mayor habilidad. Hay que mencionar que el proceso de diseño involucra emplear herramientas tanto de software, hardware y conocimientos de sistemas digitales.

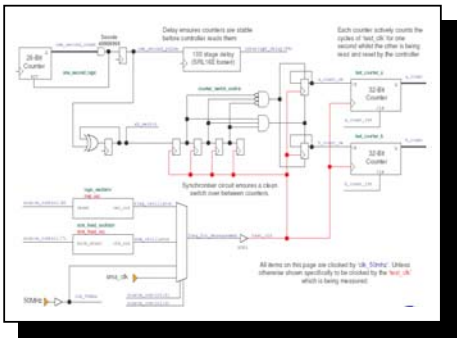
Las materias referentes al aprendizaje del diseño de sistemas digitales es preponderante en una carrera de Ingeniería Electrónica, de aquí la motivación a realizar el presente trabajo en este sentido, se desarrollan cuatro ejemplos de diseños digitales que se imparten en la materia "Diseño de sistemas digitales", con la idea de que pueda ser un complemento de la misma.

OBJETIVOS:

- Realizar la síntesis e implementación de diseños de sistemas digitales basados en los conceptos de la electrónica digital moderna y de lenguajes descriptivos.

- Mostrar el uso de las herramientas de de diseño actuales en el desarrollo de sistemas digitales.

- Efectuar los pasos en el proceso de diseño, desde la concepción hasta su desarrollo.



1

Conceptos generales

1.1 Reseña

Existe una diversidad de lenguaje de descripción de circuitos electrónicos y sistemas digitales muy amplia, sin embargo dos de ellos han tenido un nivel de desarrollo notable que los han hecho más populares y más socorridos para la descripción de circuitos y son el lenguaje VHDL y Verilog. El lenguaje VHDL, que responde a las siglas VHSIC HDL ("*Very High Speed Integrated Circuits, Hardware Description Language"*), y es ratificado por el Instituto para la Ingeniería Eléctrica y Electrónica (IEEE, en 1987) en la norma IEEE-1076. Aunque en este sentido el Verilog cumple las propuestas anteriormente anunciadas, el VHDL se impone como lenguaje estándar de diseño. Los HDL's son lenguajes alfanuméricos comprensibles para describir circuitos electrónicos en sentido amplio. En primer lugar veremos cuál ha sido la aportación de los HDL's en la metodología clásica de diseño.

Los lenguajes de descripción de hardware (HDL's) permiten modelar sistemas digitales completos. Mediante diferentes herramientas de software estos modelos pueden luego sintetizarse para implementarlos como circuitos reales. La utilización de HDL's para sintetizar sistemas digitales y la utilización de PLD's permiten crear prototipos funcionales en plazos relativamente cortos. Esto hace que todo el proceso de desarrollo de un sistema digital sea mucho más simple y rápido en comparación con metodologías clásicas (desarrollo con discretos sobre PCB's o el diseño de circuitos integrados).

Los modelos de hardware usando HDL's pueden ser estructurales, de comportamiento o una mezcla de estos dos. A nivel estructural se describe la interconexión y jerarquía entre componentes. A nivel de comportamiento de hardware se describe la respuesta entrada/salida de un componente. El comportamiento de un sistema puede modelarse a distintos niveles de abstracción o detalle: algoritmos y comportamiento general, nivel de transferencia de registros, nivel de compuertas, etc. El tipo de modelo más usado para síntesis es el denominado RTL ("*Register Transfer Level*"), o de nivel de transferencia de registros. Existen herramientas que permiten sintetizar circuitos a partir de modelos de abstracción más elevados, pero en general lo hacen llevando el diseño a un nivel de descripción como RTL antes de sintetizarlo.

La utilización de HDL's para síntesis puede tener como objetivo la creación de un circuito integrado de propósito específico (ASIC) o la implementación del circuito en alguna lógica programable (PLD). Independientemente del objetivo, no todas las construcciones posibles de los lenguajes de descripción de hardware pueden sintetizarse y transformarse en circuitos. Esto puede deberse a las limitaciones de las herramientas utilizadas o a que el circuito descrito por el modelo VHDL no puede implementarse físicamente. En general el diseñador debe seguir ciertas pautas de modelado, que dependerán de su objetivo y de las herramientas que utiliza, para que el modelo pueda sintetizarse. Esto es aún más importante para lograr implementaciones óptimas sobre la arquitectura para la que se está diseñando el circuito.

Cuando se diseña con lógicas programables, cualquiera sea el método usado para diseñar el circuito (HDL's, esquemáticos, etc.), el proceso desde la definición del circuito por el desarrollador hasta tenerlo funcionando sobre un PLD implica varios pasos intermedios y en general utiliza una variedad de herramientas. A este proceso se lo denomina ciclo o flujo de diseño.

1.2 Flujo de Diseño.

Durante el proceso de creación de un sistema digital desde el código fuente (esquemáticos, VHDL, etc.) a la implementación en un PLD hay varios pasos intermedios. Para cada uno de estos pasos se utilizan herramientas de software diferentes que pueden o no estar integradas bajo un ambiente de desarrollo. En muchos casos las herramientas utilizadas en cada paso del diseño son provistas por diferentes empresas. La Figura 1-1 muestra el ciclo de diseño típico para lógicas programables. Dependiendo de las herramientas utilizadas, este ciclo puede tener variaciones o las tareas llamarse con otros nombres. A continuación se describe cada uno de los pasos del ciclo de diseño. Se agrega su denominación en inglés entre paréntesis, ya que estos son los términos que se encontrarán en las herramientas de desarrollo.

- Descripción del Diseño: este es el paso en el que se describe el diseño, muchas veces usando un lenguaje de descripción de hardware como el VHDL. Muchas herramientas permiten ingresar el diseño no solo como HDL's sino también como un diagrama esquemático o estructural, una representación gráfica de una máquina de estados o una tabla de entrada-salida. Estas herramientas simplifican en gran medida el diseño y simplifican mucho la tarea del diseñador. El código HDL puede ingresarse utilizando cualquier editor de texto, pero se recomienda uno que tenga coloreado automático de Sintaxis ("*syntax highlighting*") ya que ayuda y hace más fácil esta etapa.
- Generación o Traducción (*Generate, Translate*): Este paso tiene sentido cuando el diseño se hace mediante algunas de los métodos mencionados anteriormente en vez de en VHDL. En este paso se traducen todos los módulos a VHDL.

Para los módulos ingresados como VHDL, las herramientas generalmente hacen una copia a una librería interna. En esta etapa se hace un análisis del VHDL para verificar la sintaxis y semántica de los módulos.

También se hace una elaboración de los archivos, que consiste en replicar los componentes que se utilizan más de una vez en el diseño para hacer copias únicas y definir un conexionado adecuado.

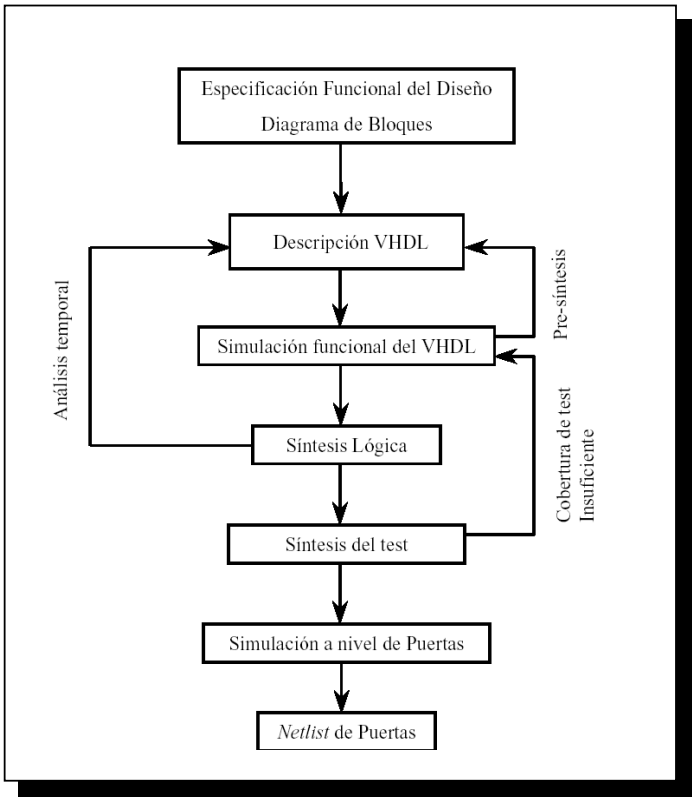


Fig. 1-1 Flujo de diseño para lógicas programables

- Compilado (*Compile*): Los simuladores actuales compilan el código VHDL a un formato que permite una simulación más rápida y eficaz. Esto se hace en este paso.

- Simulación y verificación: En este paso se simula el comportamiento del diseño y se evalúa su comportamiento. La simulación puede hacerse en tres etapas diferentes del diseño. La primera es sobre el código VHDL original para verificar el correcto funcionamiento del diseño. La segunda es después de sintetizar el circuito, y se simulan la implementación real sobre el PLD, ya sea con o sin la anotación de tiempos. La tercer etapa en la cual se puede simular el diseño es después de la Ubicación e Interconexión. Esta es la más exacta y la más engorrosa y lenta, ya que incluye la información final lógica y temporal el diseño sobre el PLD.

- Síntesis (*Synthesis*): En este paso se traduce el VHDL original a su implementación con lógica digital, utilizando los componentes específicos del PLD que va a utilizarse. Esta traducción puede llegar hasta el nivel más básico de elementos lógicos (CLB´s, LUT`s, FF´s) o hasta un nivel superior, en el que el diseño se

presenta en módulos básicos estándar provistos en una librería por el proveedor del PLD.

- Ubicación e Interconexión (*Place and Route*): El PLD está compuesto por muchos bloques idénticos, como se presentó en las secciones anteriores. En este paso, cada componente del diseño sintetizado se ubica dentro del PLD específico. También se interconectan los componentes entre sí y con los pines de entrada/salida.

- Tareas Adicionales: Las tareas adicionales dependen del fabricante y las herramientas. Puede definirse la interconexión de las con los pines físicos del PLD ingresar condiciones de entorno físico para guiar a la herramienta de Ubicación e Interconexión, seleccionar áreas del PLD para ubicar los bloques lógicos, etc.

- Anotación de Retardos: Como en todo circuito digital, las señales tendrán un retardo de propagación que influirá sobre su comportamiento. Con estos retardos puede anotarse el diseño compilado para una simulación que incluya información de temporizado mas cercana a la implementación real. Una vez sintetizado el diseño, puede hacerse una estimación de los retardos de propagación que habrá entre las señales. Después de la ubicación e interconexión, el cálculo de retardos es mucho más exacto.

- Generación de Binarios: Después de la Ubicación e Interconexión se genera algún archivo ya sea para poder utilizar el sistema en un diseño más complejo o para programar un PLD.

- Configuración de PLD: Con el archivo binario generado puede configurarse directamente un PLD a través de alguna de las opciones de configuración. Estas opciones dependerán de las herramientas y del PLD que se esté utilizando.

- Programación de Memoria (PROM): Muchas FPGA no pueden configurarse de manera permanente y requieren algún tipo de memoria para leer la configuración cuando se les aplica tensión de alimentación. En la etapa de producción deben configurarse memorias PROM y conectarlas a la FPGA en el circuito impreso.

- Verificación (automática): Una vez integrada la FPGA con su programación al sistema debe hacerse una verificación para controlar que el diseño sobre en la FPGA funciona bien (las FPGA's pueden tener fallas internas) y que la FPGA se integra bien al sistema en el que está. Pueden usarse técnicas y herramientas de verificación automáticas para evaluar si el dispositivo y el diseño están funcionando como debieran.

En la figura 1-2, las flechas de líneas punteadas que vuelven al comienzo indican las iteraciones que muchas veces se dan al trabajar en un diseño nuevo.

Después de simularlo o configurar un PLD, pueden descubrirse fallas o haber cambios de requerimientos que obligan al diseñador a volver y modificar la descripción del diseño.

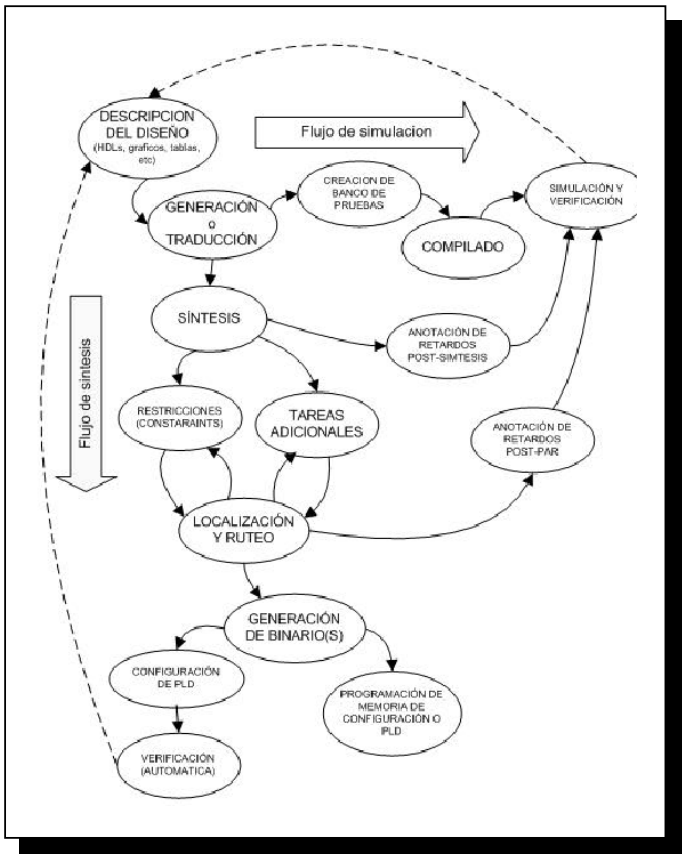


Fig. 1-2 Ruta en el proceso de diseño

1.3 Diseño Digital con FPGA's

Para poder atacar un problema complejo (sistema) la mejor manera que tenemos es dividirlo. De esta manera se pueden atacar problemas de menor complejidad y mas fáciles de analizar. Este es el camino que generalmente usamos para diseñar un sistema digital. Las decisiones que se tomen al principio para dividir el problema pueden afectar el resultado final.

En algunos casos debemos volver al principio y replantear la arquitectura después de haber resuelto el problema. La experiencia y conocimientos sobre el problema, las posibles soluciones y las tecnologías disponibles ayudarán a que la decisión que se tome sea la más acertada.

Un sistema puede estar compuesto por varios subsistemas, no todos ellos digitales o electrónicos. La primera decisión será entonces que partes del sistema se implementarán como subsistemas digitales. En el sistema digital, los siguientes pasos se siguen en cada nivel de la jerarquía del diseño, cada vez con mayor nivel de detalle.

La especificación es muy importante pare definir bien los límites de lo que se quiere fabricar. A partir de la especificación se puede definir una arquitectura con los

diferentes componentes que implementan cada función del sistema. Para el diseño se debe definir el funcionamiento de cada uno de esos componentes.

Dominio Comportamiento: Describe lo que hace un sistema (funcionamiento) Nivel Algoritmo: Describe el comportamiento del sistema como operaciones sobre las entradas para producir las salidas deseadas sin detallar los cambios en el tiempo ni las señales internas.

Nivel RTL: Describe el comportamiento de todas las señales (entradas, internas y salidas) en el tiempo (para cada ciclo de reloj en sistemas sincrónicos) Dominio Estructural: Describe al sistema como una interconexión de componentes (esquemático)

Nivel P-M-S (*Processor Memory Switch*): Describe la estructura como la interconexión de elementos de procesamiento, memoria e interconexión, sin mayores detalles de implementación.

Nivel Registros: Describe la estructura como la interconexión de registros y lógica de interconexión.

Dominio Físico: Describe la implementación física del sistema (plano).

Para lograr tiempo más rápidos de diseño, menos errores, y mayor productividad, lo ideal sería poder trabajar al máximo nivel de abstracción posible (algoritmos), sin entrar en los detalles de bajo nivel. Desafortunadamente, esto no es siempre posible con la tecnología existente (pero esta cambia permanentemente). El nivel de abstracción dependerá entonces de la aplicación y restricciones sobre el diseño.
Si se debe diseñar un integrado *full-custom* analógico es probable que se trabaje al nivel de ecuaciones diferenciales, al nivel de transistores y haciendo el *layout* manual. Si se diseña un integrado digital con bloques predefinidos (IP, Cores) y adquiridos a terceros se trabajará a un nivel de abstracción mucho mayor (*floorplanning*, RTL). En el ciclo de diseño con FPGA's, gran parte del aspecto físico ha sido solucionado por el fabricante. En general se trabaja al nivel de transferencia de registros con algunas herramientas que ayudan a diseñar a niveles de algoritmo.

1.4 Metodología General de Diseño Digital

Al desarrollar cualquier sistema digital es importante seguir ciertas pautas de trabajo y tener en cuenta factores muy diversos para que el diseño pueda terminarse a tiempo y funcione correctamente. A medida que los diseños se hacen más complejos, la necesidad de seguir un método ordenado para lograr buenos resultados se hace más importante. Los lenguajes de descripción de hardware, si se utilizan correctamente, pueden utilizarse en varios pasos del proceso de desarrollo, no solo para diseñar sino también para especificar y documentar el sistema que se quiere desarrollar.

Las grandes capacidades de los PLD's y herramientas de diseño disponibles permiten que los diseños implementados sobre FPGA's sean cada vez más complejos. En muchos casos varias personas pueden estar trabajando sobre el mismo producto, incluso en localidades separadas. Para poder atacar el problema del diseño de sistemas

digitales complejos (ya sea para desarrollos sobre FPGA's, ASIC's o PCB's) es importante tener una metodología de trabajo que permita planificar y ordenar el trabajo.

La "Metodología Universal de Diseño" define pasos que permiten:

- Diseñar un dispositivo libre de defectos de manufactura, que funciona de manera adecuada y se integra con el sistema.
- Diseñar el dispositivo de manera eficiente, sin malgastar recursos ni tiempo.
- Planificar el diseño de manera eficiente, crear un cronograma razonable y asignar los recursos necesarios para las diferentes tareas de manera ordenada.

En muchos casos el proceso es no lineal. Si se descubren fallas o problemas en las especificaciones, se debe volver a iterar sobre los pasos anteriores para corregirlos.
El ciclo comienza con un conjunto de requerimientos para la fabricación de un dispositivo o sistema. Estos requerimientos pueden venir de un cliente, de otro grupo de trabajo dentro de la misma empresa o del mismo grupo de trabajo que necesita desarrollar una parte de un sistema más grande.

1.5 Especificación y Diseño

Una especificación permite que todas las personas involucradas en un proyecto comprendan cual es el dispositivo que se va a desarrollar. Las personas que forman parte de un proyecto incluyen no solo a los desarrolladores, sino también a clientes, gerencia, personal de otras áreas de la empresa, etc. Las especificaciones deberían describir la solución de manera de cumplir con los requerimientos que se piden para el dispositivo.

Al haber una especificación formal, las bases sobre las que trabajar en un diseño quedan preestablecidas y se minimizan los errores por diferencias de apreciación o entendimiento entre los participantes. Una especificación debería comprender los siguientes puntos:

- Diagrama en bloques del sistema externo, que muestra como y donde encaja el dispositivo dentro del sistema completo.
- Diagrama en bloques interno que muestra los principales bloques funcionales.
- Descripción de las entradas/salidas, incluyendo interfaces lógicas, eléctricas y protocolos de comunicación.
- Estimaciones de tiempos que se deben cumplir, incluyendo tiempos de "*setup*" y "*hold*" para las entradas/salidas y frecuencias de reloj.
- Estimación de la complejidad y/o magnitud del dispositivo, dado en número de compuertas equivalentes o número de circuitos integrados necesarios.
- Especificación física del dispositivo. Tamaño, empaquetamiento, conectores, etc.
- Estimación del consumo de potencia del dispositivo.
- Precio estimado del dispositivo.
- Procedimientos de verificación y validación para el dispositivo.

Después de escribir las especificaciones es importante hacer una revisión con todos los miembros del equipo. De esta revisión podrán surgir cosas que no se tuvieron en cuenta individualmente y que produzcan modificaciones.

La especificación también incluye la metodología de verificación del dispositivo. Estas muchas veces se dejan para el final del proyecto y no se definen ni llevan a cabo de manera adecuada.

La especificación es un documento activo, que se modifica de acuerdo en los cambios de requerimientos y a medida que se tiene más información sobre el proyecto. Una vez que se escribe la especificación se puede utilizar para seleccionar componentes y tecnologías que se utilizarán para el proyecto.

El diseño deberá hacerse siguiendo métodos aceptados y confiables. El proceso de diseño es en general un ciclo, e incluye varios pasos intermedios.

1.6 Verificación

La verificación engloba varios pasos menores, y al revisar pueden surgir cosas que obligan a volver atrás hacia pasos anteriores. Dependiendo del dispositivo y tecnología utilizada, pero en general sigue los siguientes pasos:

- Simulación: es en general un proceso continuo, ya que al simular se pueden encontrar problemas que hacen volver sobre el diseño y hacer cambios. Las simulación se hacen sobre pequeñas partes del sistema y sobre el sistema completo. Se debe llevar a cabo una simulación funcional, pero también puede incluir simulación de temporizado, consumo de potencia y otros parámetros.
- Revisión: En este paso se revisan los resultados de la simulación y se analiza el comportamiento del dispositivo. Es importante que participen ingenieros externos al proyecto y personas que conozcan el sistema en su totalidad.
- Implementación Física: Una vez que se ha aceptado el diseño se lleva a cabo la implementación física final del dispositivo.
- Verificación Formal: En este paso se verifica la implementación física para asegurarse que su funcionamiento coincide con las simulación hechas anteriormente. En este paso se deben también evaluar los tiempos, consumo de potencia y cualquier otro parámetro de importancia.

1.7 Pasos Finales

Si todos los pasos se siguieron correctamente la revisión final debería ser solo una formalidad. Se verifica que el dispositivo está listo para ser entregado o integrado al sistema. La integración y verificación en el contexto del sistema general es muy importante. Si los pasos se siguieron correctamente, cualquier modificación que surja de esta integración será en general pequeña y no requerirá grandes cambios. Cualquier problema o falla que se encuentre debe ser documentarse y analizada para poder corregirla en una próxima versión del dispositivo y evitarla en el futuro.

2

Descripción de la tarjeta

La Tarjeta Spartan 3E Starter Kit board, es una conveniente tarjeta de desarrollo para aplicaciones de procesamiento embebido, el centro de desarrollo de esta tarjeta es un dispositivo FPGA. Tiene las siguientes características:

2.1 Características especificas de la tarjeta Spartan 3E Starter Kit:

- Configuración paralela de Flash NOR
- Configuración Flash SPI
- Desarrollo Embebido
- Procesador RISC Embebido de 32 bits -Microblaze-.
- Controlador Embebido de 8 bits –Picoblaze-.
- Interfaces a memoria DDR.

2.2 Componentes y características:

- Un FPGA Xilinx Spartan 3E XC3S500E
- 232 pin de usuario para Entrada/Salida
- Mas de 10,000 celdas lógicas
- Plataforma Flash del tipo PROM de 4 Mbits Xilinx.
- Una CPLD de 64 macroceldas Xilinx
- 64 MBytes de DDR SDRAM x 16 interface de datos a 100 MHz.
- 16 Mbyte de flash paralela NOR INTEL
- Jumpers de configuración del FPGA
- 16 Mbits de SPI serial Flash STMicro
- Pantalla LCD 16 caracteres x 2 líneas
- Puerto de teclado o mouse PS/2.
- Puerto de pantalla VGA
- 10/100 Ethernet Phy
- Dos puertos de 9 pin RS-232
- Un oscilador de reloj de 50 MHz
- Un conector de expansión HIROSE FX2
- 3 conectores de expansión de 6 pin digilent.

2.3 Hardware que compone la tarjeta *Spartan 3E Starter kit*.

Switches, Botones y perillas.

Switchs deslizables

La Spartan 3E contiene 4 switches deslizables. Estos switches están ubicados en la parte baja derecha de la tarjeta y están etiquetados como SW3 a SW0.

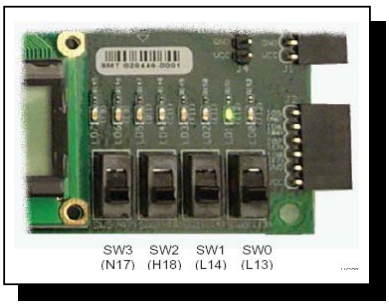


Fig. 2-1 Los cuatro switches deslizables.

Operación

Cuando el switch esta en ON o arriba, el mismo se conecta a 3.3 V. o lógica alta. Cuando el switch esta abajo o en posición OFF se conecta a tierra o lógica baja. Estos switches tienen aproximadamente 2 ms de rebote por accionamiento mecánico.

Switches de "push- button"

Localización y etiquetas.

La tarjeta contiene cuatro push- buttons de contacto momentáneo como se muestra en la figura 2-2. Los botones están localizados en la parte izquierda baja de la tarjeta y están etiquetados como: BTN_NORTH, BTN_EAST, BTN_SOUTH, y BTN_WEST. Los pines del FPGA que se conectan a los push- button aparecen entre paréntesis y están asociados a un UCF*.

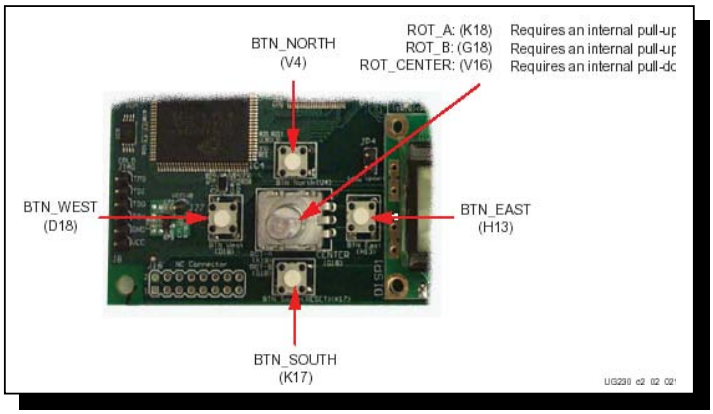


Fig. 2-2 Ubicación de los 4 push – button's y switch rotatorio.

Operación:

Presionando un push- button se conecta al pin del FPGA asociado a 3.3 V., como se observa en la figura 2-3. Utilizando una resistencia de *pull- down* dentro de la tarjeta genera un estado bajo cuando el botón no es presionado.

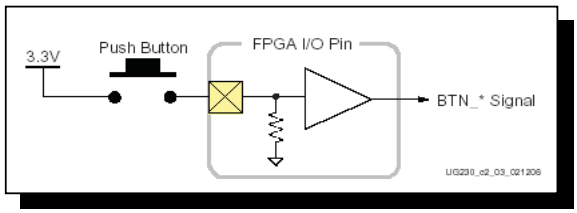


Fig. 2-3 Un push – button requiere un resistencia de Pull – Down en un pin de entrada de la FPGA.

Push- Button Rotatorio

Localización y etiquetas

El push- button rotatorio esta ubicado al centro de los cuatro push- button's individuales, mostrado en la fig. 2-2. El switch produce tres salidas. Dos salidas del *encoder* del eje que son ROT_A y ROT_B. El push- button del centro es el ROT_CENTER.

Operación

El push button rotatorio integra dos funciones distintas: El switch del eje rotatorio y los valores de salida cuando esta girando el eje.

LEDS Discretos

La tarjeta Spartan 3E tiene 8 leds de montaje superficial individuales localizados cerca de los switches deslizables como se muestra en la figura 2-4. Los leds están etiquetados como Led 7 a Led 0.

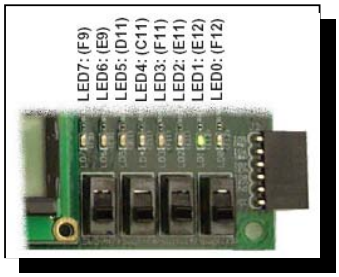


Fig. 2-4 Los 8 leds discretos.

Cada led tiene un pin conectado a tierra y el otro pin a un pin asociado en la tarjeta a través de una resistencia limitadora de corriente de 390 Ω.

Fuentes de reloj

La Spartan-3E Starter Kit board soporta tres Fuentes primarias de entrada de reloj, todas las cuales están ubicadas debajo del logo de Xilinx cerca del logo Spartan 3E, como se observa en la figura 2-5. La tarjeta incluye:

- Un reloj oscilador de 50 MHz embebido en la tarjeta.
- Un reloj que puede ser suministrado por una fuente externa vía un conector estilo SMA. Alternativamente, la Spartan 3E puede generar señales de reloj u otras señales de alta velocidad en el conector SMA.
- Opcionalmente se puede colocar un oscilador de 8 pines en el socket suministrado.

2.4 Conexionado de reloj

Cada una de las entradas de reloj se conectan a un buffer global de entradas en el bank 0 de entradas / salidas. Como se muestra en la tabla 2-1, cada una de las fuentes de reloj se conecta óptimamente a una asociada DCM.

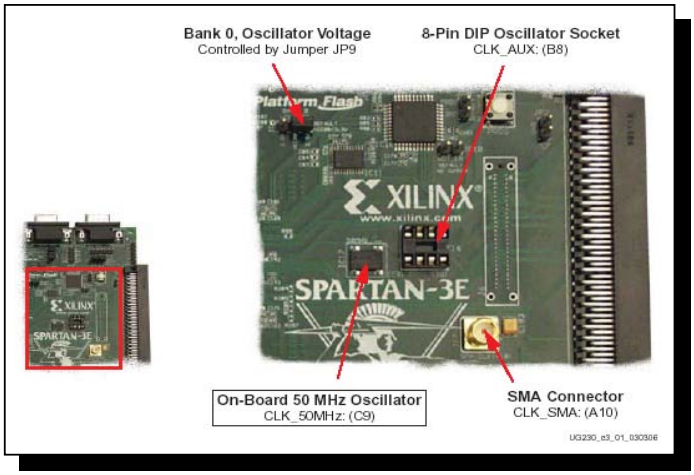


Fig. 2-5 Entradas de reloj disponibles.

Clock Input	FPGA Pin	Global Buffer	Associated DCM
CLK_50MHZ	C9	GCLK10	DCM_X0Y1
CLK_AUX	B8	GCLK8	DCM_X0Y1
CLK_SMA	A10	GCLK7	DCM_X1Y1

Tabla 2-1 Entradas de reloj y sus asociados buffer´s globales y DCM´s.

2.5 Control de voltaje.

El voltaje para todos los pines de I/O en el FPGA es controlado por el jumper JP9. En consecuencia, las fuentes de reloj están también controladas por el JP9. Por default el JP9 esta conectado a 3.3 V, puede también configurarse para que brinde un voltaje de 2.5 V.

Oscilador embebido de 50 MHz.

La tarjeta incluye un oscilador de 50 MHz con una salida de 40% a 60% de *duty cicle*. Limitaciones UCF (*User constraint Files* – Archivos de limitación de usuario -).

Las fuentes de entradas de reloj requieren de dos tipos de restricciones. La colocación de restricciones define las asignaciones de pin de I/O y los estándares de I/O. Las limitaciones del periodo OPCIONES DE CONFIGURACION DEL FPGA. Estas definen el periodo de reloj – y en consecuencia la frecuencia de reloj – y el ciclo de trabajo de la señal de reloj de entrada.

2.6 Opciones de configuración del FPGA

La tarjeta spartan 3E Starter Kit soporta una variedad de configuraciones del FPGA (Véase fig. 2-6):

- Descargar diseños a la tarjeta Spartan 3E vía JTAG, utilizando la interfaz USB. La lógica USB- JTAG proporciona además programación en el sistema para la plataforma FLASH PROM y el CPLD XC2C64A.
- Programar los 4 Mbit de plataforma serial Flash PROM de Xilinx.
- Programar los 16 Mbit de SPI Flash PROM de ST Microelectronics.
- Programar los 128 Mbits de NOR Flash PROM de Intel.

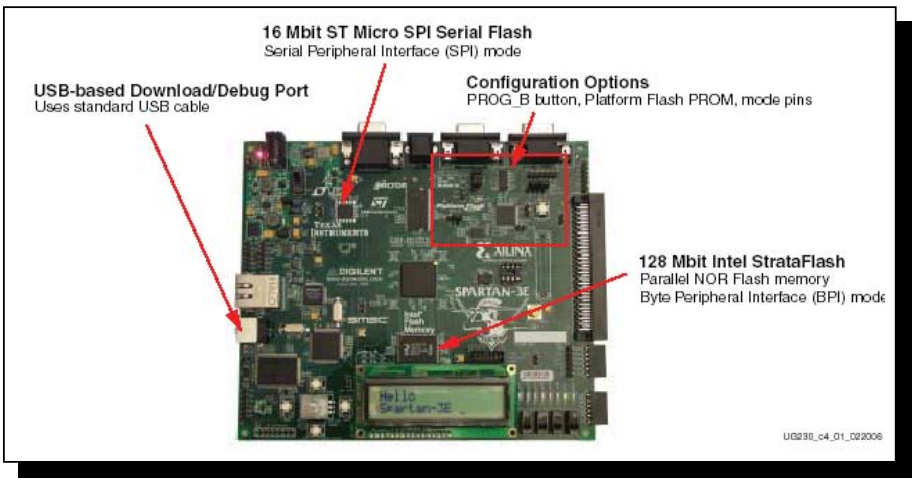


Fig. 2-6 Opciones de configuración del FPGA Spartan 3E Starter kit.

Los *jumpers* del modo de configuración determinan que modo de configuración del FPGA cuando es aplicado el voltaje de suministro o siempre que el botón de PROG sea presionado.

El led del pin DONE enciende cuando la configuración del FPGA finaliza satisfactoriamente. Presionando el botón PROG se forza al FPGA a restablecer el proceso de configuración.

Los 4 MBits de Flash PROM de Xilinx proporcionan una fácil configuración JTAG-Programable para almacenamiento del FPGA.

Estos *jumpers;* que se emplean para dicha configuración se muestran en la figura 2-7.

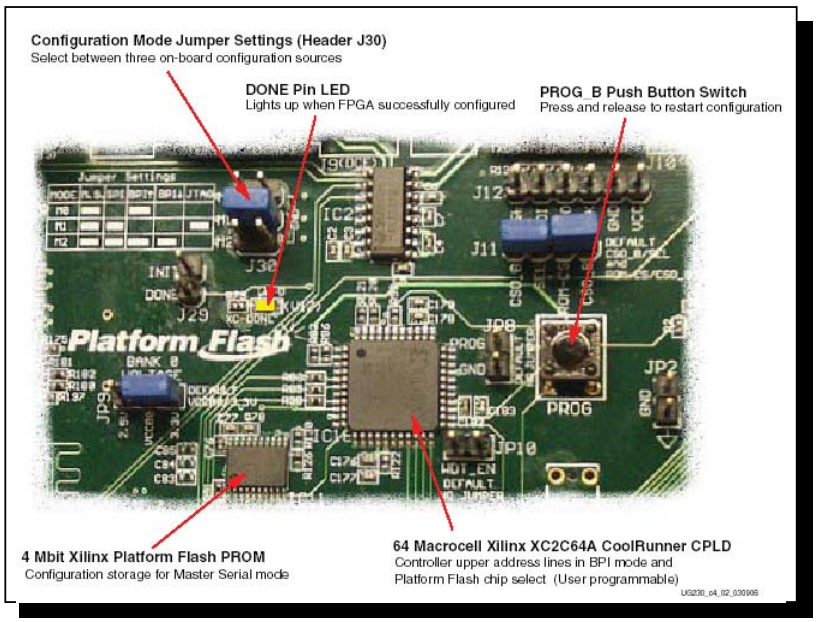


Fig. 2-7 Vista detallada del área de configuración.

Jumpers de Modo de configuración.

En la figura 2-8, se muestra el arreglo del modo de configuración, el cual esta dado por J30. Insertando o removiendo los *jumpers* individuales seleccionamos el modo de configuración del FPGA y su asociada configuración de fuente de memoria.

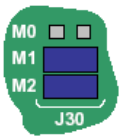
Modo de Configuracion	Modo de Pins M2:M1:M0	Fuente de Imagen de config. FPGA	Ajuste de Jumper
Master Serial	0:0:0	Platform Flash PROM	
SPI	1:1:0	SPI Serial Flash PROM starting at address 0	

Fig. 2-8 Modo de configuración de acuerdo a la posición de jumpers

Push Button PROG

El push – button PROG forza al FPGA a reconfigurarse desde la configuración seleccionada de la fuente de memoria. Presionando y liberando este botón restablecemos el proceso de configuración del FPGA en cualquier momento.

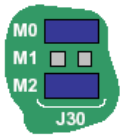
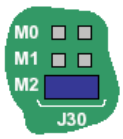
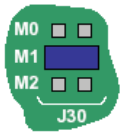
BPI Up	0:1:0	StrataFlash parallel Flash PROM, starting at address 0 and incrementing through address space. The CPLD controls address lines A[24:20] during BPI configuration.	
BPI Down	0:1:1	StrataFlash parallel Flash PROM, starting at address 0x1FF_FFFF and decrementing through address space. The CPLD controls address lines A[24:20] during BPI configuration.	
JTAG	0:1:0	Downloaded from host via USB-JTAG port	

Fig. 2-8 (Cont.) Modo de configuración de acuerdo a la posición de jumpers

2.7 Programación del FPGA, o plataforma Flash PROM vía USB.

La tarjeta Spartan 3E incluye una lógica de programación embebida basada en USB, así como una terminación USB tipo B. Vía un cable conectado al host del PC, el software de programación IMPACT programa directamente el FPGA, la plataforma Flash PROM, o bien el CPLD.

Conexión del cable USB

El kit incluye un cable estándar tipo A / Tipo B, como el que se presenta en la figura 2-9.

Después de instalar el software de Xilinx, conectar del lado del conector tipo B a la tarjeta Spartan 3E Starter kit, como se muestra en la figura. Cuando la tarjeta es encendida, el sistema operativo (Windows) puede entonces reconocer e instalar el controlador asociado.

Una vez que el controlador del cable USB es instalado satisfactoriamente, y la tarjeta es correctamente conectada a la PC, un led color verde encenderá indicando una correcta conexión.

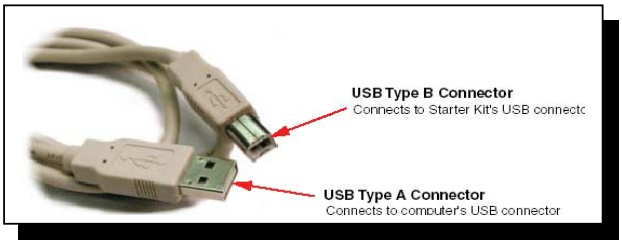


Fig. 2-9 Cable estándar USB tipo A/ tipo B.

2.8 Pantalla de caracteres LCD

Una de las características prominentes de la Spartan 3E Starter Kit es que contiene un displací de cristal liquido de 2 líneas x 16 caracteres. El FPGA controla el LCD vía un interfaz de datos de 4 bits como se muestra en la figura 2-10. Aunque el LCD soporta un interfaz de datos de 8 bits, la tarjeta utiliza solo 4 bits de interfaz de datos para permitir que sea compatible con otras tarjetas de desarrollo de Xilinx.

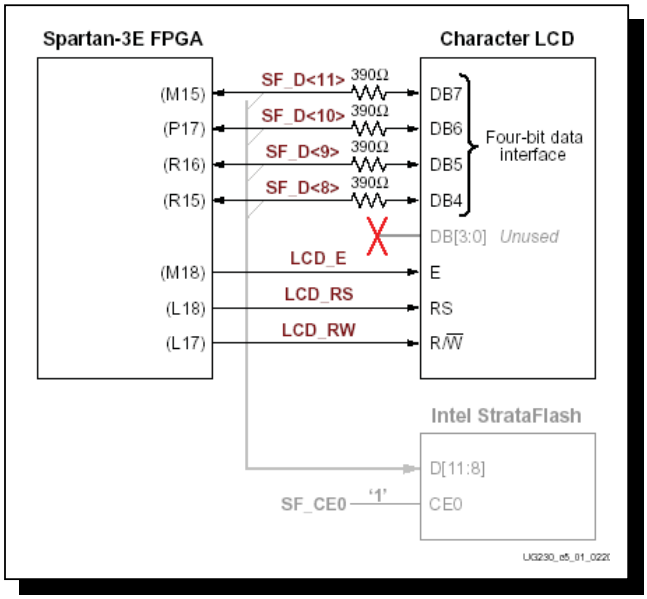


Fig.2-10 Interface de caracteres LCD.

Una vez que se domina, el LCD es una vía práctica de presentar información utilizando el estándar ASCII y caracteres tradicionales. Sin embargo este presentador no es tan rápido. Comparado con el reloj de 50 MHz el display es lento. Un procesador Picoblaze eficientiza el control del tiempo de presentación del display.

Señales de interfaz del LCD de caracteres

La tabla 2-2, nos muestra las señales de interfaz del LCD.

Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	

Tabla 2-2. Señales de interfaz del LCD.

2.9 Compatibilidad de voltaje

El LCD de caracteres es encendido por un voltaje de 5 V. Las señales de I/O son suministradas por 3.3 V. Sin embargo, los niveles de salida del FPGA son reconocidos como un nivel lógico bajo o alto por el LCD. El controlador del LCD acepta niveles de señal de 5 V TTL y la salida de 3.3 V proporcionada por el FPGA requiere de un nivel de voltaje de 5V.

Los resistores de 390 Ω en serie en la línea de datos previenen una sobre tensión en el FPGA y en los pines de I/O cuando el LC D maneja un valor de lógica alta. El LCD maneja línea de datos cuando el LCD_RW esta en alto. Muchas aplicaciones manejan al LCD como dispositivo de solo escritura, ya que nunca leen desde el display.

2.10 Puerto de Pantalla VGA

La tarjeta Spartan 3E Starter Kit incluye un puerto para una pantalla VGA vía un conector DB15. Se pueden conectar la mayoría de pantallas VGA o monitores de pantalla plana mediante un cable de monitor estándar.

La Spartan 3E maneja directamente las 5 señales de VGA por medio de resistores (Véase fig. 2-11). Cada línea de color tiene un resistor en serie. Los resistores en serie, en combinación con unas terminaciones de 75 Ω están construidos en el cable VGA. Las señales VGA_HSYNC y VGA_VSYNC utilizan niveles de I/O LVTTL o LVCMOS33. El manejo de las señales VGA_RED, VGA_GREEN, y VGA_BLUE en alto o en bajo generan los 8 colores presentados en la tabla 2-3.

2.11 Puertos serial RS-232

La tarjeta Spartan 3E Starter Kit tiene dos puertos serial RS-232: Un conector hembra DB9 DCE y un conector macho DTE como se muestra en la figura 2-12.

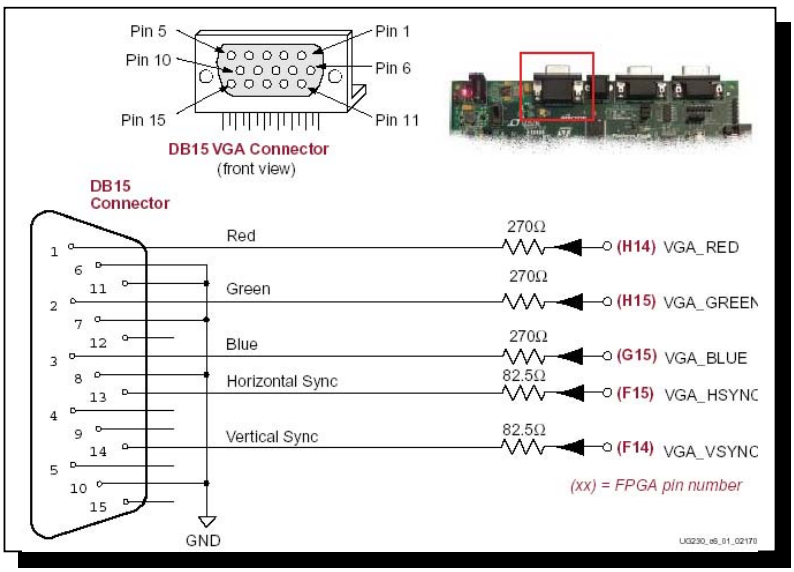


Fig. 2-11 Conector VGA, su ubicación.

VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Tabla 2-3. Códigos de color del display de 3 bits.

El puerto tipo DCE se conecta a la mayoría de los equipos PC o estaciones de trabajo mediante un cable serial. Se utiliza el conector tipo DCE para controlar algún otro periférico como un modem u otro dispositivo serie.

2.12 Puerto de teclado y Mouse PS/2

La tarjeta Spartan 3E Starter Kit incluye un puerto de Mouse y teclado PS/2 y el estándar mini DIN de 6 pines, etiquetado como J14 en la tarjeta, en la figura 2-13, se muestra la ubicación de este conector y de las señales que maneja el mismo. Solo los pines 1 y 5 del conector son empleados por el FPGA.

2.12 Puerto de teclado y Mouse PS/2

La tarjeta Spartan 3E Starter Kit incluye un puerto de Mouse y teclado PS/2 y el estándar mini DIN de 6 pines.

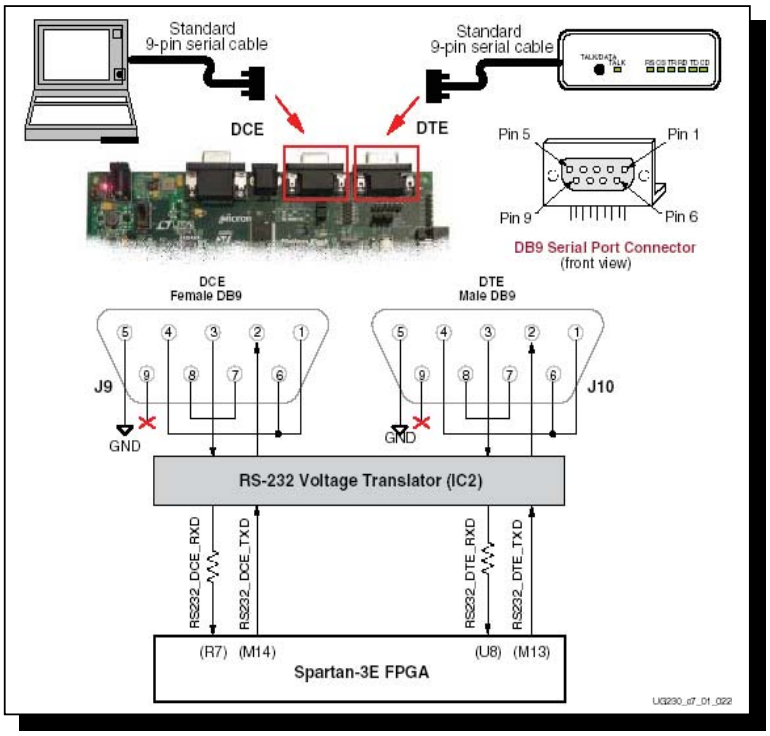


Fig. 2-12 Puertos serial RS- 232.

Etiquetado como J14 en la tarjeta, en la figura 2-13(a) y (b), se muestra la ubicación de este conector y de las señales que maneja el mismo. Solo los pines 1 y 5 del conector son empleados por el FPGA.

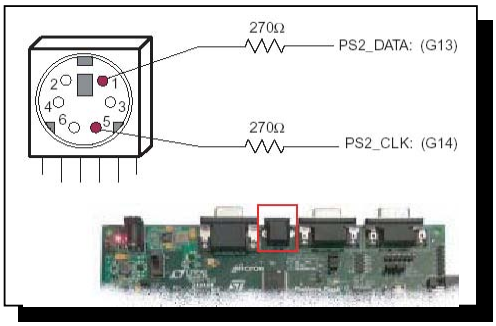


Fig. 2-13a Ubicación del conector PS/2.

Tanto el teclado como el Mouse de la PC utilizan los dos cables del bus serial PS/2 para comunicarse con la tarjeta Spartan 3E. El PS/2 incluye tanto la señal de reloj como de datos. Tanto el Mouse como el teclado manejan el bus con idénticos tiempos de señal y ambos usan una palabra de 12 bits que incluyen señal de arranque, paro, y bit de paridad.

Sin embargo, el paquete de datos esta organizado de diferente manera para una Mouse y para un teclado. Además, la interfaz del teclado dispone de transferencia de datos bidireccional.

PS/2 DIN Pin	Signal	FPGA Pin
1	DATA (PS2_DATA)	G13
2	Reserved	G13
3	GND	GND
4	+5V	—
5	CLK (PS2_CLK)	G14
6	Reserved	G13

Fig. 2-13b Ubicación del conector PS/2 y sus señales.

Los tiempos del bus PS/2 aparecen en la tabla 2-4. Las señales de reloj y de datos son manipuladas únicamente cuando ocurre la transferencia de datos. El tiempo define los requerimientos de señal para la comunicación del Mouse a la tarjeta Spartan y la comunicación bidireccional del teclado.

Symbol	Parameter	Min	Max
T_{CK}	Clock High or Low Time	30 µs	50 µs
T_{SU}	Data-to-clock Setup Time	5 µs	25 µs
T_{HLD}	Clock-to-data Hold Time	5 µs	25 µs

Tabla 2-4. Tiempos del bus PS/2.

2.13 Convertidor Digital –analógico

La tarjeta Spartan 3E incluye un convertidor serial digital – analógico, de cuatro canales, con compatibilidad SPI. El DAC es un dispositivo con tecnología lineal LTC2624 con una resolución de 12 bits sin signo. Las cuatro salidas del DAC aparecen en el conector J5, el cual utiliza un formato de modulo periférico Digilent de 6 pines. El DAC y el conector están localizados cerca del conector RJ-45 Ethernet, como se muestra en la figura 2-14.

Comunicación SPI

El FPGA utiliza un interfaz periférico serial (SPI) que comunican los valores digitales de cada uno de los cuatro canales del DAC. El bus SPI es un *full- duplex*, síncrono, que emplea un interfaz simple de 4 hilos.

Un bus maestro – el del FPGA en este caso – maneja el bus de señal de reloj (SPI_SCK) y transmite datos en forma serial (SPI_MOSI) para seleccionar un bus esclavo – El DAC en este caso - . De forma similar, el bus esclavo proporciona datos seriales (SPI_MISO) que retornan al bus maestro (véase figura 2-15).

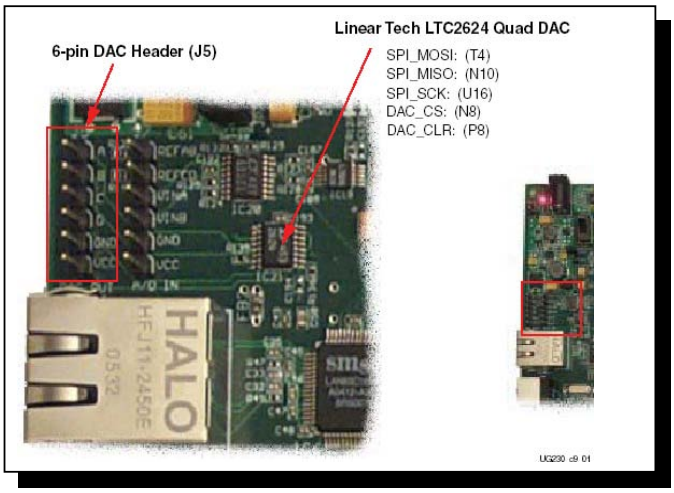


Fig. 2-14 Convertidor Digital – Analógico y su conector asociado.

2.14 Circuito de captura analógica.

La tarjeta Spartan 3E incluye un circuito de captura analógica de dos canales, que consiste de un preamplificador escalable programable y de un convertidor analógico – digital (ADC), como se muestra en la figura 2-16. Las entradas analógicas están dispuestas en el conector J7.

El circuito de captura analógica consiste de un preamplificador programable de tecnología Lineal LTC6912-1 que escala la señal analógica de entrada desde el cabezal J7. La salida del preamplificador se conecta a un ADC LTC1407A-1.

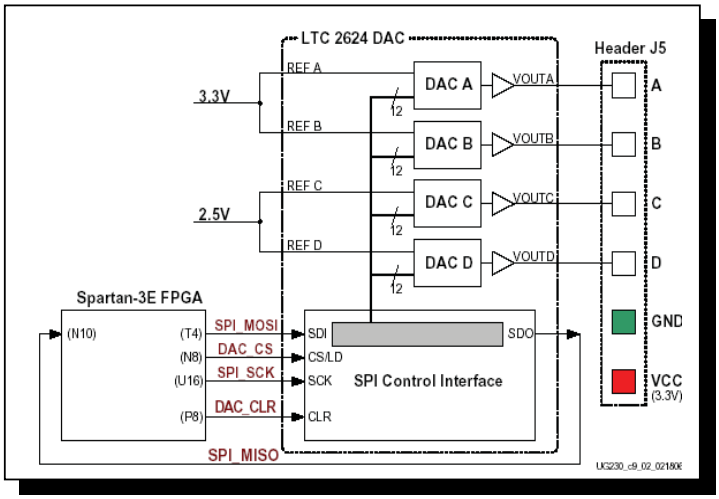


Fig. 2-15 Esquema de conexión Digital – analógica.

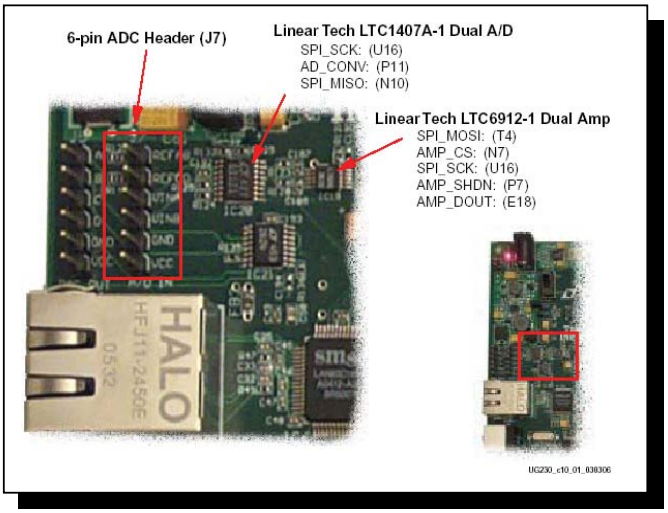


Fig. 2-16 Circuito de captura analógico de 2 canales.

Ambos, tanto el preamplificador como el ADC son programados serialmente o controlados por el FPGA, un esquema del circuito se muestra en la figura 2-17.

2.15 Salidas digitales a entradas analógicas

El circuito de captura analógica convierte el voltaje analógico en VINA o VINB convirtiendo este a una representación digital de 14 bits, expresado por la ecuación 2-1.

$$D[13{:}0] = GAIN \times \frac{(V_{IN} - 1.65V)}{1.25V} \times 8192$$

Ecuación 2-1.Ganancia de entrada analógica.

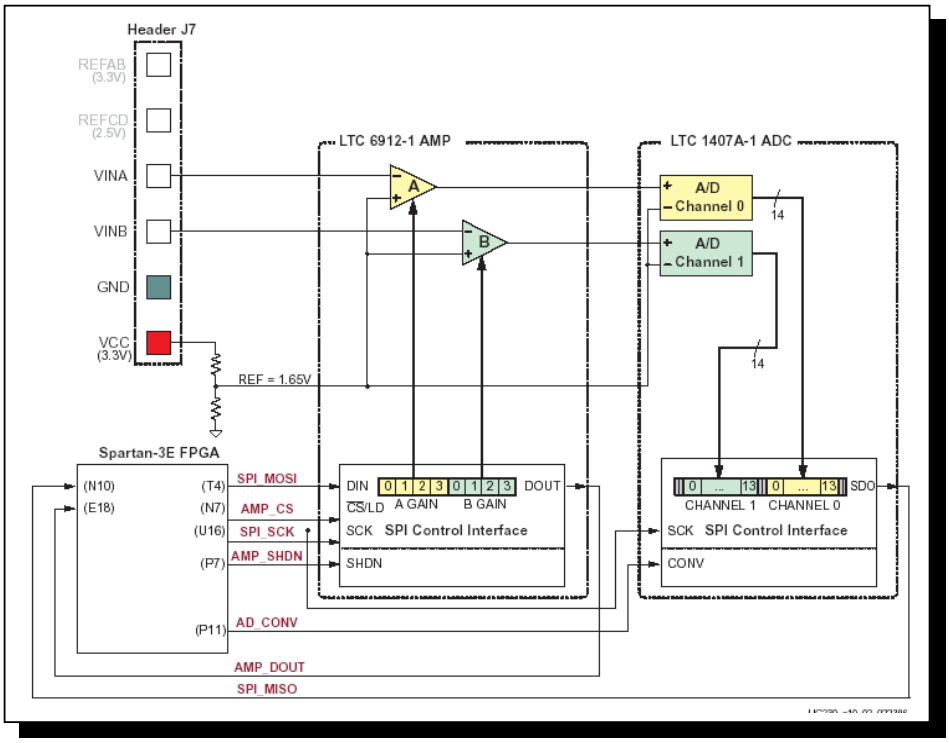


Fig. 2-17 Circuito de captura analógica.

La ganancia es el ajuste de corriente seleccionada en el preamplificador programable. Los diversos ajustes disponibles para la ganancia y los voltajes permisibles aplicables a las entradas VINA o VINB se muestran en la figura. La referencia de voltaje para el amplificador y el ADC es de 1.65 V, generado por medio de un divisor de voltaje. En consecuencia, 1.65 V es restado del voltaje de entrada VINA o VINB. El rango máximo del ADC es de ±1.25 V, centrado alrededor de la referencia de voltaje 1.65 V.

2.16 Intel StrataFlash Parallel NOR Flash PROM.

Como se muestra en la siguiente figura, la tarjeta Spartan 3E incluye 128 MBits (16 MByte) de Intel StrataFlash parallel NOR Flash PROM. Como se muestra en la figura 2-18, algunas de las conexiones están compartidas con otros componentes de la tarjeta.

La PROM Strataflash proporciona diferentes funciones:

- Almacena una configuración simple de FPGA en el dispositivo Strataflash.
- Almacena dos configuraciones distintas de FPGA en el dispositivo Stratflash y puede conmutar de manera dinámica entre las dos configuraciones utilizando la característica Multiboot de FPGA de la Spartan 3E.
- Almacena y ejecuta código del procesador Microblaze directamente del dispositivo Strataflash.
- Almacena datos no volátiles de la FPGA.

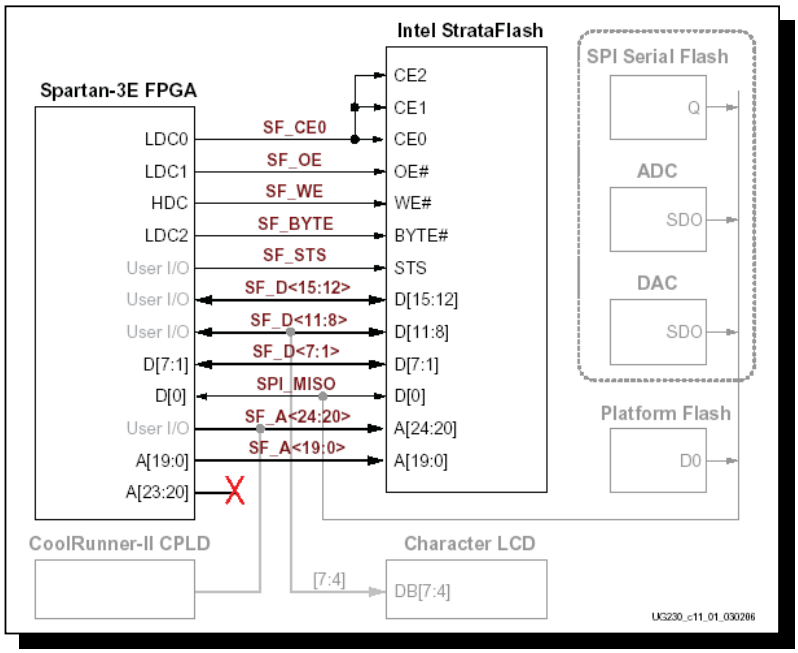


Fig. 2-18 Conexiones del FPGA a la memoria Flash Intel Strataflash.

2.17 Flash Serial SPI

La Spartan 3E incluye memoria Flash Serial SPI de 16 MBits (ST microelectronics) muy útil en una variedad de aplicaciones. La SPI flash proporciona un medio alternativo de configurar la FPGA.

La SPI flash esta también disponible para la FPGA después de la configuración para una variedad de propósitos como son:

- Almacenamiento de datos no volátiles
- Almacenamiento para códigos de identificación, números de serie, direcciones de IP, etc.
- Almacenamiento de código para el procesador Microblaze que puede ser compartido por una DDR SRAM.

Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA→SPI	Serial data: Master Output, Slave Input
SPI_MISO	N10	FPGA←SPI	Serial data: Master Input, Slave Output
SPI_SCK	U16	FPGA→SPI	Clock
SPI_SS_B	U3	FPGA→SPI	Asynchronous, active-Low slave select input

Tabla 2-5. Señales de interfaz Flash SPI.

Configuración para la Flash SPI

Si se desea configurar la FPGA para la Flash SPI debe ser puesto adecuadamente la selección de pines de modo y la Flash SPI debe contener una configuración de imagen valida, se muestra en la figura 2-19 la ubicación de jumpers.

Pines de selección de modo

Se coloca la configuración de modo del FPGA en modo SPI, como se muestra en la figura 2-20.

2.18 DDR SRAM

La Spartan 3E Starter Kit contiene una memoria DDR SDRAM de 512 MBit de Micron Technology con un interfaz de datos de 16 bits, como se muestra en la figura 2-21. Todos los pines de interfaz del DDR SRAM se conectan al BANK 3 de I/O del FPGA.

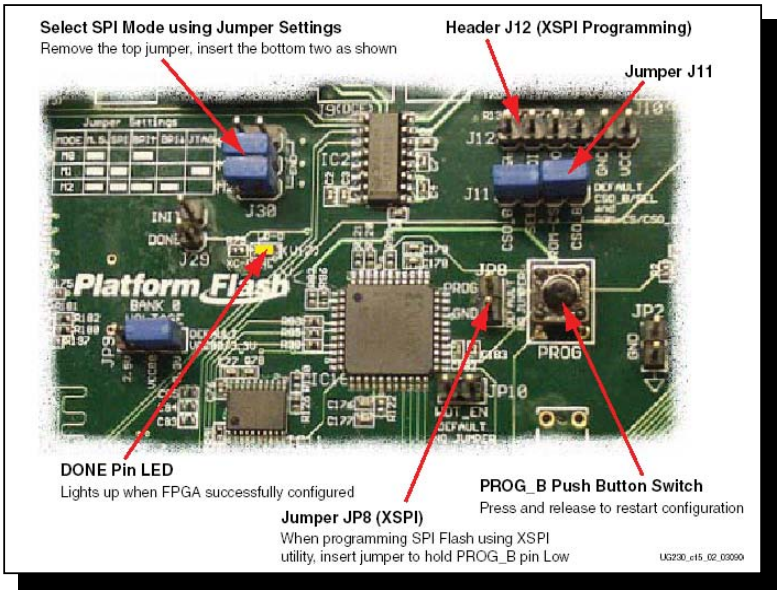


Fig. 2-19 Opciones de configuración para el modo SPI.

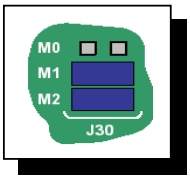


Fig. 2-20 Pin de ajuste de modo SPI.

2.18 DDR SRAM

La Spartan 3E Starter Kit contiene una memoria DDR SDRAM de 512 MBit de Micron Technology con un interfaz de datos de 16 bits, como se muestra en la figura 2-21. Todos los pines de interfaz del DDR SRAM se conectan al BANK 3 de I/O del

FPGA. Ambos son alimentados por un voltaje de suministro de 2.5 V regulado por un LTC3412.

2.19 Interfaz Física Ethernet 10/100

La tarjeta Spartan 3E Starter Kit contiene también una interfaz de capa física Ethernet 10/100 (PHY) y un conector RJ-45, como se muestra en la figura 2-22. Con un controlador de acceso al medio Ethernet (MAC) implementado en la FPGA, la tarjeta se puede conectar de manera opcional a una red ethernet estándar. Todos los ciclos de reloj están controlados por un oscilador de cristal de 25 MHz.

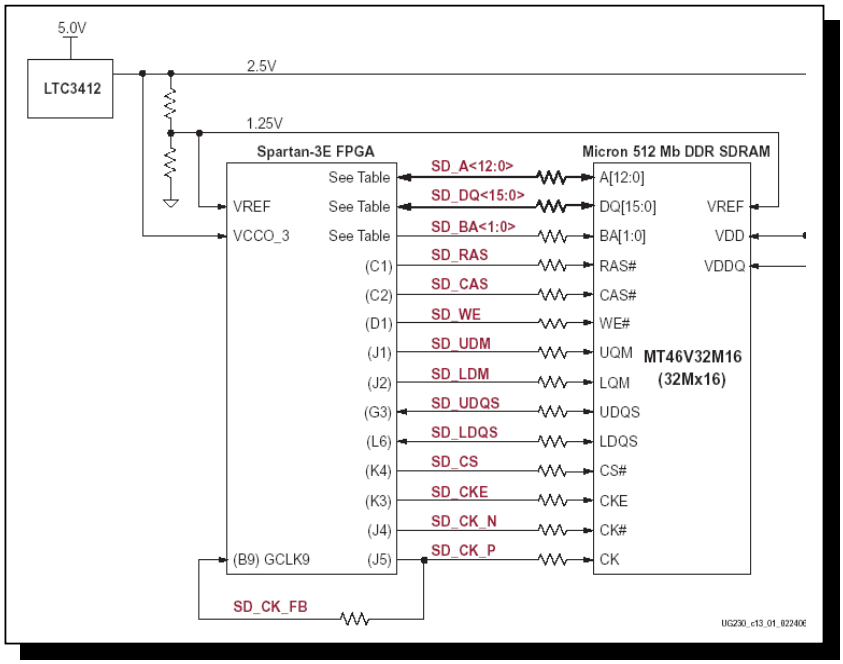


Fig. 2-21 Interfaz de FPGA a la DDR SRAM

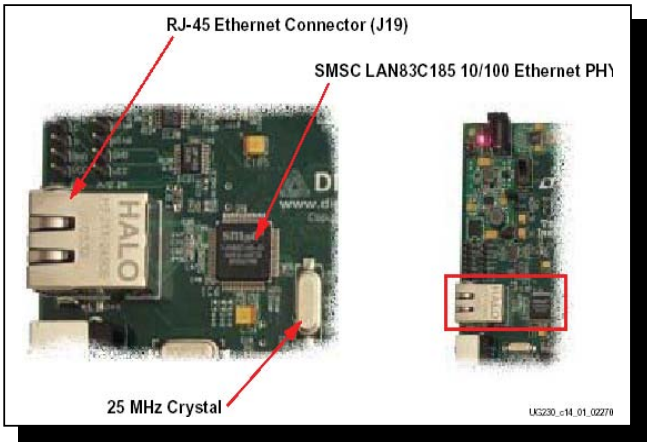


Fig. 2-22 Ethernet 10/100 con conector RJ-45.

Conexiones Ethernet PHY

La FPGA se conecta al Interfaz PHY Ethernet utilizando un interfaz de medio independiente (MII).

2.20 Conectores de expansión

La Spartan 3E Starter kit suministra una variedad de conectores de expansión para una interfaz sencilla y flexible con otros componentes fuera de la tarjeta. La tarjeta incluye los siguientes conectores de expansión de I/O.

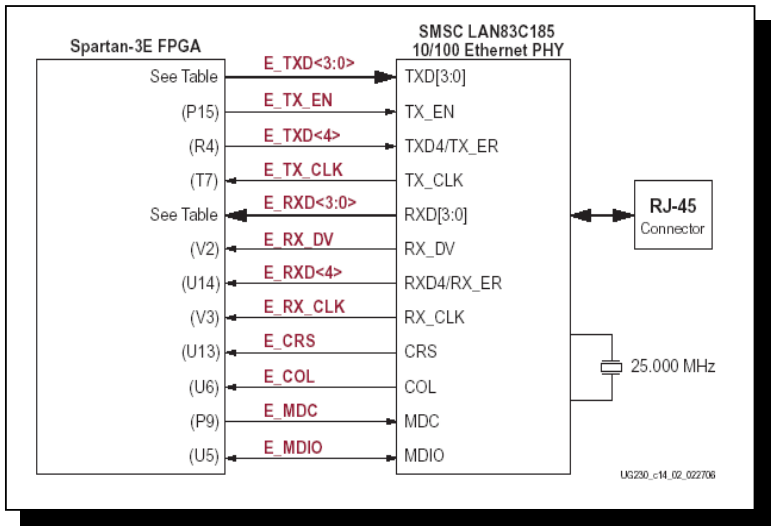


Fig. 2-23. Interfaz del FPGA al Ethernet Phy.

Un conector Hirose de 100 pines con 43 pines de usuario de I/O asociados al FPGA, incluyendo hasta 15 pares diferenciales de I/O LVDS y dos pares de entrada simple. Tres conexiones de modulo periférico de 6 pines. Plataforma para un probador Agilent o Tektronix.

La ubicación de estos conectores esta mostrada en la figura 2-24.

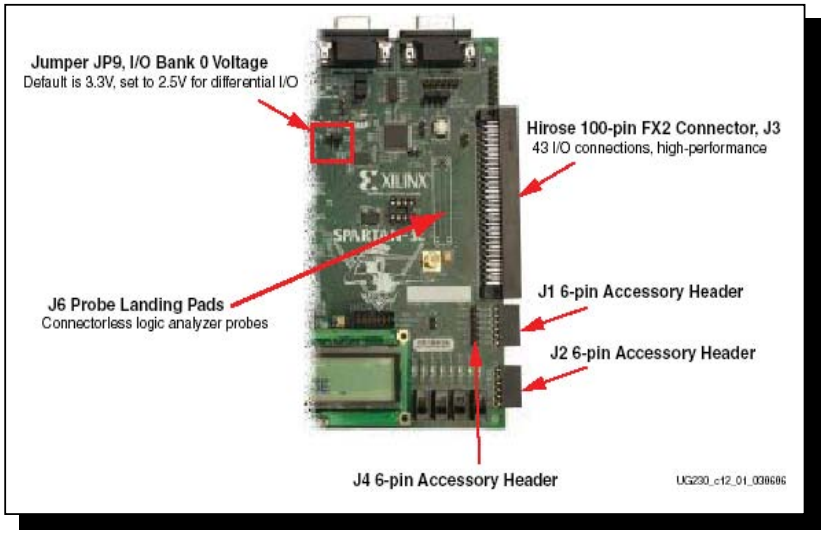


Fig. 2-24 Conectores de expansión.

Conector FX2 Hirose de 100 pines (J3)

Un conector de 100 pines esta ubicado del costado derecho de la tarjeta. Se trata de un conector Hirose FX2-100P-1.27DS.

En la figura 2-25, se muestra como del conector FX2, cinco de sus pines son I/O bidireccionales capaces de enviar o recibir señales. Cinco pines, FX2_IP<38:35> y FX2_IP<40> son pines únicamente de entrada al FPGA.

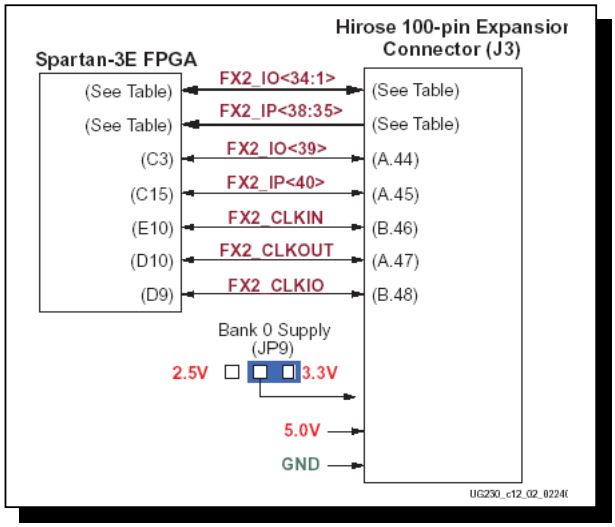


Fig. 2-25 Conexiones del dispositivo FPGA hacia el conector Hirose.

Accesorio conector de 6 pines

El accesorio conector de 6 pines proporciona una sencilla expansión de interfaz de I/O utilizando los diferentes módulos periféricos digilent. La ubicación de estos conectores en la tarjeta se muestra en la figura 2-26.

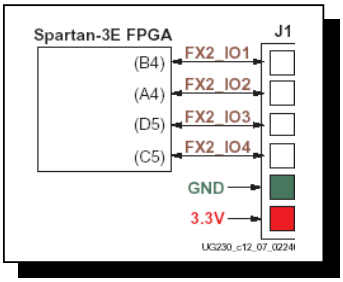


Fig. 2-26 Conexiones del conector J1

El conector J1 es un conector de 6 pines con un socket tipo hembra. Cuatro pines del FPGA se conectan al conector J1, los cuales son FX2_IO<4:1>. Estas cuatro señales son también compartidas con el conector Hirose FX2. La tarjeta suministra en la parte baja del socket un voltaje de 3.3 V.

De manera similar al conector J1, también existen otros dos conectores de conexión fácil, que son los conectores J2 y J4 que funcionan de igual forma al J1, cambiando únicamente la dirección asignada.

3

Lenguajes de descripción: VHDL, Verilog

3.1 Historia del VHDL

Las siglas VHDL provienen de "VHSIC Hardware Description Lenguaje" y a su vez VHSIC quiere decir "*Very High Speed Integrated Circuit*". O sea que VHDL significa lenguaje de descripción de hardware para circuitos integrados de alta velocidad. Sus orígenes datan de la década de 1980. El departamento de defensa de Estados Unidos y el IEEE patrocinaron un programa de desarrollo para un lenguaje con el que se pudieran modelar circuitos integrados de alta complejidad. Un lenguaje de estas características permitiría describir los circuitos para su documentación y además modelarlos y evaluarlos mediante simulaciónes antes de incurrir en los grandes gastos de fabricación.

El VHDL nació entonces como un lenguaje de modelado y documentación de sistemas electrónicos digitales. El lenguaje se estandarizó mediante el estándar 1076 del IEEE en 1987 (VHDL-87). Este estándar fue extendido y modificado en 1993 (VHDL-93) y 2002 (VHDL-2002). En la actualidad, VHDL se utiliza no solo para modelar circuitos electrónicos sino también para crear, o sintetizar, nuevos circuitos. La capacidad de sintetizar circuitos a partir de los modelos en VHDL (u otro lenguaje de descripción de hardware) surgió después de la creación del lenguaje, con la aparición de herramientas que traducen los modelos VHDL a circuitos reales.

3.2 Historia de Verilog

El lenguaje Verilog fue desarrollado por *Gateway Design Automation* en 1984. En 1988 *Synopsis* presentó la primera herramienta de síntesis basada en Verilog. Más tarde *Cadence Design Systems* adquirió *Gateway Design Automation* y luego "abrió" el lenguaje para que otros proveedores pudieran desarrollar herramientas de simulación y síntesis utilizando Verilog. En 1995 el lenguaje Verilog se transformó en un estándar del IEEE. Mientras que la sintaxis del VHDL es parecido a los lenguajes de programación Ada y Pascal, el Verilog proviene del C y tiene una sintaxis mas parecida a este.

3.3 Otros

En la actualidad los dos lenguajes mencionados, VHDL y Verilog, son los más utilizados para la síntesis automática de hardware. Hay mucho esfuerzo de investigación y desarrollo (e incluso algunas herramientas comerciales) que aceptan la descripción de algoritmos en otros lenguajes con un mayor nivel de abstracción.

Muchos vendedores de FPGA's proveen herramientas o "*toolboxes*" que se integran a lenguajes de modelado como Matlab y Simulink. Una vez probado un algoritmo con estas herramientas se genera de manera automática el hardware necesario para implementarlo. Una de las líneas de desarrollo principales es la de poder describir los sistemas con un nivel de abstracción mayor al RTL en un único lenguaje Después se especificarían las restricciones necesarias para los resultados y las herramientas decidirían que parte conviene implementar en hardware y que partes en software.

En general todas las herramientas actuales, aunque permiten describir o modelar los sistemas con un lenguaje mas abstracto que el RTL terminan generando alguna versión de código RTL antes de sintetizar un circuito.

3.4 Lenguaje de Descripción VHDL

En 1983, IBM, Intermetrics y Texas Instruments empezaron a trabajar en el desarrollo de un lenguaje de diseño que permitiera la estandarización, facilitando con ello, el mantenimiento de los diseños y la depuración de los algoritmos, para ello el IEEE propuso su estándar en 1984.

Tras varias versiones llevadas a cabo con la colaboración de la industria y de las universidades, que constituyeron a posteriori etapas intermedias en el desarrollo del lenguaje, el IEEE publicó en diciembre de 1987 el estándar IEEE std_1076-1987 que constituyó el punto firme de partida de lo que después de cinco años sería ratificado como VHDL.

Esta doble influencia, tanto de la empresa como de la universidad, hizo que el estándar asumido fuera un compromiso intermedio entre los lenguajes que ya habían desarrollado previamente los fabricantes, de manera que éste quedó como ensamblado y por consiguiente un tanto limitado en su facilidad de utilización haciendo dificultosa su total comprensión. Este hecho se ha visto incluso ahondado en su revisión de 1993.

Pero esta deficiencia se ve altamente recompensada por la disponibilidad pública, y la seguridad que le otorga el verse revisada y sometida a mantenimiento por el IEEE. La independencia en la metodología de diseño, su capacidad descriptiva en múltiples dominios y niveles de abstracción, su versatilidad para la descripción de sistemas complejos, su posibilidad de reutilización y en definitiva la independencia de que goza con respecto de los fabricantes, han hecho que VHDL se convierta con el paso del tiempo en el lenguaje de descripción de hardware por excelencia. En 1987 el estándar IEEE 1076.1 para VHDL fue aprobado.

Las siguientes convenciones aplican a los diseños VHDL:

- VHDL no es sensible a mayúsculas/minúsculas.

- Dos guiones "- -" indican el inicio de un comentario.
- Los nombres pueden usar caracteres alfanuméricos y el guión bajo "_"
- Los nombres deben comenzar con un carácter alfabético.
- No se pueden usar dos guiones bajos continuos o como último carácter.
- No se permiten espacios en los nombres.
- Los objetos nombrados deben ser únicos, es decir, no se puede tener una señal llamada A y un bus A (7 *downto* 0).

La siguiente tabla 3-1, es una lista de palabras VHDL reservadas:

component	else	file	end	exit	loop	for	map
Configuration	abs	function	mod	Generate	Body	Buffer	nand
constant	generic	after	new	group	guarded	access	alias
disconnect	all	and	architecture	impure	inertial	assert	attribute
downto	if	in	begin	inout	is	block	array
elsif	label	library	linkage	literal	bus	case	not
entity	nor	range	record	register	To	null	next
rem	reject	Transport	on	Then	Type	of	or
report	Until	srl	subtype	Wait	package	port	open
return	Use	severety	Unaffected	process	Xnor	out	xor
rol	Units	shared	While	When	others	With	postponed
ror	Variable	sra	procedure	pure	signal	sla	select

Tabla 3-1. Palabras reservadas en VHDl.

Una descripción en VHDL está formada por:

- Declaración de librerías,
- Terminales de entrada y salida (entidad),
- Arquitectura.

3.5 Sentencias concurrentes y secuenciales

Para iniciarnos correctamente en el aprendizaje y manejo de VHDL es importante que comprendamos desde un principio la diferencia entre concurrente y secuencial.

El concepto de concurrencia, se ve claramente graficado en los circuitos electrónicos donde los componentes se encuentran siempre activos, existiendo una asociación intrínseca, entre todos los eventos del circuito; ello hace posible el hecho de que si se da algún cambio en una parte del mismo, se produce una variación (en algunos casos casi instantánea) de otras señales.

Este comportamiento de los circuitos reales obliga a que VHDL soporte estructuras específicas para el modelado y diseño de este tipo de especificaciones de tiempos y concurrencias en el cambio de las distintas señales digitales de los diseños. Por el contrario, las asignaciones secuenciales, son más bien propias de los SDL ("*soft design language*") en los que la programación tiene un flujo natural secuencial, siendo

propio de este tipo de eventos las sentencias *case, if, while, loop,* etc. más propias de estas sintaxis.

Las construcciones concurrentes del lenguaje son usadas dentro de estructuras concurrentes, por ejemplo una arquitectura tiene una naturaleza eminentemente concurrente (es decir que está activo todo el tiempo), mientras que el cuerpo de un *process* es en principio eminentemente secuencial.

La asignación de eventos secuenciales dentro de una estructura concurrente se ejecutará de forma concurrente, es decir, al mismo tiempo que las demás sentencias.

VHDL soporta con este motivo, tres tipos de objetos, las variables, las constantes y las señales. Como las variables y las señales pueden variar su valor mientras ejecutamos un programa, serán éstas las encargadas de almacenar dichos datos, asimismo serán los portadores de la información. Únicamente las señales pueden tener la connotación de globalidad dentro de un programa, es decir, que pueden ser empleadas dentro de cualquier parte del programa a diferencia de las variables que solo tienen sentido en el interior de un *process* .

Sentencias secuenciales

En la mayoría de los lenguajes de descripción de software, todas las sentencias de asignación son de naturaleza secuencial. Esto significa que la ejecución del programa se llevara a cabo de arriba a abajo, es decir siguiendo el orden en el que se hayan dispuesto dichas sentencias en el programa, por ello es de vital importancia la disposición de las mismas dentro del código fuente.

VHDL lleva a cabo las asignaciones a señales dentro del cuerpo de un proceso (process) de forma secuencial, con lo que el orden en el que aparezcan las distintas asignaciones será el tenido en cuenta a la hora de la compilación. Esto hace que cuando utilicemos modelos secuenciales en VHDL, estos se comporten de forma parecida a cualquier otro lenguaje de programación como Pascal, C, etc.

Sentencia if: La construcción *if-then-else* es usada para seleccionar un conjunto de sentencias para ser ejecutadas según la evaluación de una condición o conjunto de condiciones, cuyo resultado debe ser o *true* o *false*. Su estructura es la siguiente:

```
if (condición) then
  haz una cosa;
else
  haz otra cosa diferente;
end if;
```

Si la condición entre paréntesis es verdadera, la(s) sentencia(s) secuencial(es) seguidas a la palabra *then* son ejecutadas. Si la condición entre paréntesis es falsa, la(s) sentencia(s) secuencial(es) seguidas a la palabra *else* son ejecutadas. La construcción debe ser cerrada con las palabras *end if.*

La sentencia *if-then-else* puede ser expandida para incluir la sentencia *elsif,* la cual nos permite incluir una segunda condición si no se ha cumplido la primera (la cual tiene prioridad). Su estructura es la siguiente:

```
if (condición) then
  haz una cosa;
elsif (otra condición) then
  haz otra cosa diferente;
else
  haz otra totalmente diferente;
end if;
```

Si se da la situación en la cual la primera condición es verdad ejecuta las sentencias que van después del primer *then*. Si no es verdadera la primera condición, se pasa a evaluar la segunda, y de ser esta verdad, ejecuta las sentencias que están a continuación del segundo *then.* Si ninguna de las dos es verdadera, se ejecuta lo que está detrás de la palabra *else*.

Sentencia *Case*: La sentencia case es usada para especificar una serie de acciones según el valor dado de una señal de selección. Esta sentencia es equivalente a la sentencia *with-select-when*, con la salvedad que la sentencia que nos ocupa es secuencial, no combinacional. La estructura es la siguiente:

```
case (señal a evaluar) is
  when (valor 1) => haz una cosa;
  when (valor 2) => haz otra cosa;
  ...
  when (último valor) => haz tal
cosa;
end case;
```

En el caso que la señal a evaluar (situada después del *case*) tenga el "valor 1", entonces se ejecuta "una cosa", si tiene el "valor 2", se ejecuta "otra cosa", ... y si tiene el "último valor", se ejecuta "tal cosa".

Si se da la situación en la cual la primera condición es verdad ejecuta las sentencias que van después del primer *then*. Si no es verdadera la primera condición, se pasa a evaluar la segunda, y de ser esta verdad, ejecuta las sentencias que están a continuación del segundo *then.* Si ninguna de las dos es verdadera, se ejecuta lo que está detrás de la palabra *else.*

Sentencias concurrentes

La naturaleza propia de los circuitos eléctricos obliga a VHDL a soportar un nuevo tipo de asignación de señales, que nos permita implementar este tipo de operatividad. En ella todas las asignaciones se llevan a cabo en paralelo (al mismo tiempo). En una asignación concurrente la señal que esté a la izquierda de la asignación es evaluada siempre que alguna de las señales de la derecha modifique su valor (figura 3-2).

Si las señales de entrada (situadas a la derecha), a o b, cambian su valor, las señales de salida (situadas a la izquierda), c y s, son evaluadas, de forma que su valor se verá modificado si fuese necesario.

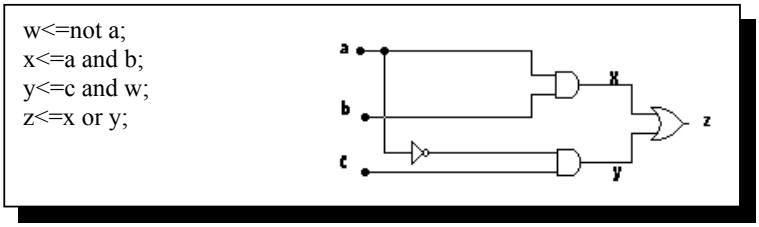


Fig. 3-2 Sentencia de asignación.

3.6 Sentencia Process

La sentencia *process* es una de las construcciones típicas de VHDL usadas para agrupar algoritmos. Esta sentencia se inicia (de forma opcional) con una etiqueta seguida de dos puntos (:), después la palabra reservada *process* y una lista de variables sensibles. La *lista sensible*, indica que señales harán que se ejecuta el proceso, es decir, qué variable(s) debe(n) cambiar para que se ejecute el proceso. Dentro de un proceso se encuentran sentencias secuenciales, no concurrentes. Esto hace que el orden de las órdenes dentro de un proceso sea importante, ya que se ejecuta una después de otra, y los posibles cambios que deba haber en las señales alteradas se producen después de evaluar todo el ciclo al completo. Esta característica define una de las particularidades de VHDL.

La estructura de un proceso es la siguiente:

etiqueta: process (var1, var2, ...) begin sentencias secuenciales end process etiqueta;	*Si cambia alguna de las variables situadas entre los paréntesis, se ejecuta el proceso.*

Siempre que queramos utilizar sentencias secuenciales se deberá usar un proceso. Como ejemplo del uso de un proceso se muestra el siguiente fragmento de código correspondiente a un flip-flop d.

contador: process (clock)
begin
  if (clock'event and clk='1') then
  q <= d;
  end if,
end process contador;
	Si hay un cambio en la señal clock (reloj), se ejecuta el process que contiene una sentencia if. Una vez acabado de evaluar todas las sentencias del proceso, se cambian todas las señales necesarias a la vez.

Es muy importante tener en cuenta dos cosas a la hora de usar un proceso respecto a las señales y las variables.

- La primera es que las variables toman instantáneamente el valor especificado y que sólo tienen sentido de existencia dentro de un proceso.

- La segunda es que las señales cambian su valor solamente al llegar al final del proceso.

3.7 Diferencias entre variable y señal.

Una de las cosas que mas resultan extrañas al programador experimentado con otros lenguajes, y que se enfrenta por primera vez a algún lenguaje de descripción de hardware, y mas concretamente al VHDL, es la diferencia entre señal y variable.

Se puede plantear en términos de analogía diciendo que una señal es como una caja con dos secciones. Una, que es la sección que se suele ver, y que es la que contiene el valor actual, y otra, separada, que contiene el valor futuro. Cuando leemos una señal estamos echando mano de la sección donde se guarda el valor actual. Cuando se le asigna algo a una señal estamos escribiendo en la sección dedicada al valor futuro. Solo cuando se acaba la ejecución de la instrucción concurrente, lo que se encuentra en la sección de valor futuro pasa a la sección de valor actual.

3.8 Como se declara una Entidad

En la declaración de entidades, se definen las entradas, salidas y tamaño de un circuito, explicitando cuales son, de qué tamaño (de 0 a n bits), modo (entrada, salida, ...) y tipo (integer, bit,...) . Las entidades pueden definir bien las entradas y salidas de un diseño más grande o las entradas y salidas de un chip directamente. La declaración de entidades es análoga al símbolo esquemático de lo que queremos implementar, el cual describe las conexiones de un componente al resto del proyecto, es decir, si hay una entrada o puerto de 8 bits, o dos salidas o puertos de 4 bits, etc.

La declaración de entidades tiene la siguiente forma:

entity circuito a is	Cabecera del programa
port(	Se indica que a continuación viene los puertos (o grupos señales) de entrada y/o salida
-- puertos de entradas -- puertos de salidas -- puertos de I/O -- puertos de buffers	*Aquí se declaran las entradas y/o salidas con la sintaxis que se verá a continuación. Las líneas empezadas por dos guiones son ignoradas por el compilador. Así mismo, recordamos que el compilador no distingue las mayúsculas de las minúsculas*
); end circuito_a;	Se indica que se ha acabado la declaración de puertos de entrada y/o salida, y que se ha acabado la entidad

Como se ha dicho, cada señal en una declaración de entidad está referida a un puerto (o grupo de señales), el cual es análogo a un(os) pin(es) del símbolo esquemático. Un puerto es un objeto de información, el cual, puede ser usado en expresiones y al cual se le pueden asignar valores.

A cada puerto se le debe asignar un nombre válido. A continuación, un ejemplo:

nombre_variable: modo tipo;	*Forma genérica de designar un puerto*
puertoa: in bit;	*El primer puerto es un bit de entrada, y su nombre es "puertoa"*
puertob: in bit_vector(0 to 7);	*El segundo puerto es un vector de 8 bits de entrada siendo el MSB el puertob(0) y el LSB el puertob(7)*
puertoc: out bit_vector(3 downto 0);	*El tercer puerto es un vector de 4 bits de salida siendo el MSB el puertoc(3) y el LSB el puertoc(0)*
puertod: buffer bit;	*El cuarto puerto es un buffer de un solo bit, cuyo nombre es "puertod"*
puertoe: inout std_logic;	*El quinto puerto es una entrada/salida del tipo estándar logic de un solo bit*

Como se puede deducir del ejemplo anterior, seguido del nombre del puerto y separado de éste por dos puntos, se debe indicar el tipo de puerto. El modo describe la dirección en la cual la información es transmitida a través del puerto: in, out, buffer e inout. Si no se especifica nada, se asume que el puerto es del modo in.

- Modo in: Un puerto es de modo in si la información correspondiente al mismo, entra a la entidad y se suele usar para relojes, entradas de control (como las típicas load, reset y enable), y para datos de entrada unidireccionales.
- Modo out: Un puerto es de modo out si la información fluye hacia fuera de la entidad. Este modo no permite realimentación ya que al declarar un puerto como out estamos indicando al compilador que el estado lógico en el que se encuentra no es leíble. Esto le da una cierta desventaja pero a cambio consume menos recursos de nuestros dispositivos lógicos programables.
- Modo buffer: Es usado para una realimentación interna ,es decir, para usar este puerto como un driver dentro de la entidad. Este modo es similar al modo out, pero además, permite la realimentación y no es bidireccional, y solo puede ser conectado directamente a una señal interna, o a un puerto de modo *buffer* de otra entidad. Una aplicación muy común de este modo es la de salida de un contador, ya que debemos saber la salida en el momento actual para determinar a salida en el momento siguiente.
- Modo inout: Es usado para señales bidireccionales, es decir, si necesitamos que por el mismo puerto fluya información tanto hacia dentro como hacia afuera de la entidad. Este modo permite la realimentación interna y puede reemplazar a cualquiera de los modos anteriores, pudiéndose usar este modo para todos los puertos, pero reduciremos la lectura posterior del código por otra persona, y reduciendo los recursos disponibles de la dispositivo.

Como se ha comentado más arriba, el lenguaje sólo admite cuatro modos para los puertos, pero puede haber tantos tipos de señales como queramos, ya que los podemos crear nosotros mismos. VHDL incorpora varios tipos de forma estándar (por haber sido creado así), pudiendo usar otros definidos en librerías normalizadas, y las creados por el usuario. La norma internacional IEEE 1076/93 define cuatro tipos nativos para VHDL como son:

- Tipo boolean: puede tomar dos valores: verdadero/true o falso/false. Un ejemplo típico es la salida de un comparador que da verdadero si los números comparados son iguales y falso si no lo son:

equal:out boolean;	*Sólo puede tomar dos valores: verdadero o falso, y es de salida (darle mas operatividad a la salida de un comparador sería superfluo)*

- Tipo *bit*: Puede tomar dos valores: 0 ó 1 (o también "*low*" o "*high*", según se prefiera). Es el tipo más usado de los nativos.
- Tipo *bit_vector*: Es un vector de bits. Debemos tener cuidado al definir el peso de los bits que lo integran, ya que según pongamos la palabra reservada *downto* o *to* estaremos diciendo que el bit más significativo es el número más alto o el más bajo del vector, respectivamente..

numero : bit_vector (0 to 7);	*En este caso el MSB es numero(0) y numero(7) el LSB*
numero : bit_vector (7 downto 0);	*En este caso el MSB es numero(7) y numero(0) el LSB*

- Tipo integer: Para manejar números enteros. Hay que advertir que el uso de enteros consume muchos recursos del dispositivo de lógica programable, siempre y cuando sea sintetizable, debido a que está prácticamente creado para la simulación.

Pero ante la necesidad de ampliar la operatividad del tipo bit, la norma IEEE 1164, definió un nuevo tipo llamado *std_logic*, *std_ulogic*, y sus derivados tales como *std_logic_vector* y *std_ulogic_vector*. Como su nombre pretende indicar, es el tipo de tipo lógico estándar, que es el más usado en la actualidad.

Como ejemplo, a continuación se incluye la declaración de una entidad correspondiente a un multiplexor de 2x1 de cuatro bits, con entrada de habilitación o *enable.* El multiplexor necesita las entradas de información, la señal de selección, la de *enable* y las salidas de información.

entity multi is port (	*Cabecera ya estudiada arriba, en la que multi es el nombre de la entidad*

enable: in bit;
selec: in bit;
in1: in bit_vector(3 downto 0);
in2: in bit_vector(3 downto 0);
out1:out bit_vector(3 downto 0)
	• *enable es un bit de entrada (suficiente para habilitar o no)*
	• *selec es otro bit de entrada, que selecciona la entrada in1 o in2, ambas de 4 bits*
	• *out1 es de salida, que lógicamente, debe ser de la misma longitud que in1 e in2*

); end multi;	*Notesé que el último puerto no lleva punto y coma al final de la línea. Si lo llevase estaría incorrecto.*

A continuación se muestra otro ejemplo correspondiente a la entidad para un comparador:

entity compa is port (	*Cabecera de la entidad, cuyo nombre es compa*
a,b: in bit_vector(3 downto 0); igual: out bit;	• ***a** y **b** son las entradas de cuatro bits* • ***igual** es la salida de un sólo bit*
); end compa;	*Se finaliza la entidad con la palabra clave end y el nombre de la misma (compa).*

Debemos recordar dos puntos más a la hora de dar el nombre a algún puerto.

- VHDL no distingue las letras mayúsculas de las minúsculas, por lo que un puerto llamado por nosotros "EnTraDA" será equivalente a otro que se llame "ENTRADA" o "entrada".
- El primer carácter de un puerto sólo puede ser una letra, nunca un número. Así mismo, no pueden contener caracteres especiales como $, %, ^, @, y dos caracteres de subrayado seguidos.

Estos dos detalles a tener en cuenta surgieron del comité que creó este lenguaje, por lo que no se debe considerar como un fallo de nuestra herramienta (WARP2), sino como una característica más del lenguaje.

3.9 Como se declara una Arquitectura

La arquitectura indica el tipo de procesado que se realiza con la información correspondiente a las señales de entrada, (declarados previamente en la entidad) para llegar a tener los puertos de salida (también declarados en la entidad). En la declaración de arquitecturas es donde reside todo el funcionamiento de un circuito, ya que es ahí donde se indica que hacer con cada entrada, para obtener la salida. Si la entidad es vista como una "caja negra", para la cual lo único importante son las entradas y las salidas, entonces, la arquitectura es el conjunto de detalles interiores de la caja negra.

La declaración de arquitecturas debe constar de las siguientes partes como mínimo, aunque suelen ser más:

architecture archpro of programa is	*Cabecera de la arquitectura. En ésta,* ***archpro*** *es un nombre cualquiera (suele empezar por "arch", aunque no es necesario) y* ***programa*** *es el nombre de una entidad existente en el mismo fichero*
-- declaración de señales y otros accesorios	*Declaraciones de apoyo, que se verán en la página siguiente*
begin	*Se da comienzo al programa*
-- núcleo del programa	*Conjunto de sentencias, bucles, procesos, funciones,... que dan operatividad al programa.*
end archpro;	*Fin del programa*

Como podemos apreciar, es una estructura muy sencilla, y que guarda alguna relación con Turbo Pascal. Las sentencias entre *begin* y *end* son las que describen el circuito, y es en lo que se centra tanto este libro electrónico como cualquier otro que trate sobre VHDL.

A continuación, se muestra el código fuente de un programa en VHDL de un multiplexor (esta es una de las múltiples formas de implementar un multiplexor en VHDL), el cual debe ir unido a la entidad expuesta en el apartado de la declaración de entidades, ya que una parte sin la otra carecen de sentido.

architecture archimulti of multi is	*Cabecera de la arquitectura. En esta ocasión el nombre de la arquitectura es archimulti, y el de la entidad es multi, la cual está*

	definida anteriormente.
-- señales	*En este programa no se necesitan señales*
begin	*Comienza al programa*

process(enable,in1,in2)
begin
if enable='0' then out1<="1111";
elsif enable='1' then
if(selec = '0') then out1<=in1;
elsif(selec = '1') then
out1<=in2;
end if;
end if;
end process;
	Sentencias que hacen que la entidad definida como multiplexor realice la función propia de su nombre. Solo hay que entender la estructura de las arquitecturas en estos momentos, por lo que este código no debe ser objeto de preocupación.

end archimulti;	*Fin del programa*

Para describir una arquitectura podremos usar cuatro estilos, teniendo cada uno, su propio nivel de abstracción. Los estilos son:

- Estilo *behavioral* o comportamiento: Este estilo se caracteriza por incluir las sentencias y órdenes típicas de un lenguaje de programación (*if, then, case,...*), sin importarnos como quedará la distribución de puertas lógicas dentro de la PLD. Es necesario un proceso al ser una estructura secuencial. El siguiente fragmento de código describe un comparador (usando una entidad descrita en este mismo tutorial) escrito con el estilo *behavioral* o de comportamiento.

architecture behavioral of compa is
begin
comp: process (a, b)
begin
if a= b then
igual<='1';
else
igual<='0';
end if;
end process comp;
end behavioral;
	Como se puede apreciar en este ejemplo se utilizan los clásicos if then else de cualquier lenguaje de programación, y además las asignaciones son secuenciales. Esto es lo que hace esta arquitectura de comportamiento o behavioral.

- Estilo *dataflow* o flujo de datos: Este estilo podremos encontrarlo de dos formas similares, pero ambas implican cómo la información será transferida de señal a señal y de la entrada a la salida sin el uso de asignaciones secuenciales, sino concurrentes. Es decir, en este estilo no se pueden usar los procesos. El comparador descrito de forma *behavioral* o de comportamiento se puede escribir usando el estilo *dataflow* de cualquiera de las dos formas siguientes:

architecture dataflow1 of compa is begin igual<='1' when (a=b) else '0'; end dataflow1;	*Esta arquitectura es del estilo dataflow porque se especifica como la información pasará a la salida sin usar sentencias secuenciales*
architecture dataflow2 of compa is begin igual<= not(a(0) xor b(0))	*Aquí de nuevo las asignaciones son concurrentes, no secuenciales.*

and not(a(1) xor b(1)) and not(a(2) xor b(2)) and not(a(3) xor b(3)); end dataflow2;	

- Estilo *structural* o estructural: En él se describe un "*netlist*" de VHDL, en los cuales los componentes son conectados y evaluados instantáneamente mediante señales. No se suele usar este estilo únicamente en una arquitectura ya que resulta muy lioso y difícil de modificar, siendo de verdadera utilidad cuando debemos crear una estructura grande y deseamos descomponerla en partes para manejarla mejor, y para hacer la simulación de cada parte más sencilla. Se suele requerir el uso de señales auxiliares, y además paquetes y librerías accesorios, lo cual, recordemos, debe estar declarado al comienzo de la entidad.

architecture struct of compa is
signal x: bit_vector(0 to 3);
begin
u0: xnor2 port map (a(0),b(0),x(0));
u1: xnor2 port map (a(1),b(1),x(1));
u2: xnor2 port map (a(2),b(2),x(2));
u3: xnor2 port map (a(3),b(3),x(3));
u4: and4 port map (x(0),x(1),x(2),x(3),igual);
end struct;
	Aquí solo se interconexionan salidas con entradas de componentes. La salida viene dada por la operatividad de los componentes, la cual no se puede saber si no conocemos el paquete del cual ha sido leída.

- Estilo mixto: Es el estilo que está compuesto en mayor o menor medida de dos o más de los estilos descritos anteriormente. Deberemos tener en cuenta que el código VHDL que escribamos no siempre va a describir una función de forma óptima, la cual no siempre va a poder ser reducida por la herramienta de compilación. Esto se traduce en un peor aprovechamiento de los recursos de las PLD's. Por lo tanto, diferentes diseños producen diferentes, aunque equivalentes, ecuaciones de diseño, pudiéndose dar, sin embargo, disposiciones diferentes de los recursos

Es habitual el usar el estilo estructural para descomponer un diseño en unidades manejables, siendo cada unidad diseñada por equipos de trabajo distintos. El estilo estructural se usa además para tener un grado de control alto sobre la síntesis.

3.10 Expresiones y operadores

La metodología de programación de un componente, basada en la descripción por comportamiento (*behavioral*), puede emplear en su diseño la mayoría de operadores que se encuentran habitualmente útiles en los SDL's (*software design languages*).

En el cuadro adjunto se puede ver una relación de los operadores predefinidos más empleados en VHDL, así mismo se aprecia que su clasificación atiende al tipo de dato que vaya a manejar.

Los operadores lógicos, pueden ser empleados con los tipos predefinidos, BIT y BOOLEAN, dándonos como resultado un valor booleano del mismo tipo que los operadores.

OPERADORES LÓGICOS	NOT, AND, OR, NAND, NOR, XOR	Tipo de operador: boolean Tipo de resultado: boolean
OPERADORES RELACIONALES	= / < <= > >=	Tipo de operador: cualquier tipo Tipo de resultado: boolean
OPERADORES ARITMÉTICOS	+ - * / ** MOD, REM, ABS	Tipo de operador: integer, real, signal tipo de resultado: integer, real,signal
OPERADOR CONCADENACIÓN	&	Tipo de operador: array tipo de resultado: array

3.11 VHDL para síntesis

La síntesis de un circuito a partir de VHDL consiste en reducir el nivel de abstracción de la descripción de un circuito hasta convertirlo en una definición puramente estructural cuyos componentes son los elementos de una determinada librería de componentes, que dependerá del circuito que se quiera realizar, la herramienta de síntesis, etc. Al final del proceso de síntesis se debe obtener un circuito que funcionalmente se comporte igual que la descripción que de el se ha hecho.

En principio, cualquier descripción en VHDL es sintetizable, no importa el nivel de abstracción que la descripción pueda tener.

Esto, que pudiera parecer sorprendente, no lo es en absoluto ya que cualquier descripción VHDL se puede simular, y si se puede simular, el propio simulador (en general un ordenador ejecutando un programa) es un circuito que funcionalmente se comporta tal y como se ha descrito. Es evidente que no será el circuito mas optimizado para realizar la tarea que se pretende, ni lo hará a la velocidad que se requiere, pero seguro que funcionalmente se comporta tal y como se ha descrito.

La complejidad del circuito resultante, y también incluso la posibilidad o no de realizar el circuito, va a depender sobre todo del nivel de abstracción inicial que la descripción tenga. En primera aproximación se puede tomar un ordenador que ejecute la simulación, teniendo entonces la síntesis. A partir de esta primera aproximación hay que ir optimizando el circuito. En realidad las herramientas de síntesis siguen una aproximación distinta, ya que de otra manera, el circuito seria algo parecido a un microprocesador cuando quizás solo se pretende implementar una puerta lógica.

La aproximación de las herramientas de síntesis consiste en, partiendo de la descripción original, reducir el nivel de abstracción hasta llegar a un nivel de descripción estructural. La síntesis es por tanto una tarea vertical entre los niveles de abstracción de un circuito.

Así, una herramienta de síntesis comenzaría por la descripción comportamental abstracta y secuencial e intentaría traducirla a un nivel de transferencia entre registros descrita con ecuaciones de conmutación. A partir de esta descripción se intentaría transformarla a una descripción estructural donde se realiza además lo que se llama el mapeado tecnológico, es decir, la descripción con los componentes de una librería especial que depende de la tecnología con la cual se quiera realizar el circuito.

3.12 Descripción de maquinas de estados.

Es muy normal, a la hora de definir hardware, realizar la descripción siguiendo la definición de una maquina de estados. Una maquina de estados esta definida por dos funciones, una calcula el estado siguiente en que se encontrara el sistema, y la otra calcula la salida. El estado siguiente se calcula, en general, en función de las entradas y del estado presente. La salida se calcula como una función del estado presente y las entradas. Normalmente hay dos tipos de maquinas de estados, unas son las de *Mealy* y las otras son las de *Moore*. Las de *Mealy* son más generales y se caracterizan porque la salida depende del estado y la entrada. Las maquinas de Moore son un caso particular de las anteriores y se caracterizan porque la salida solo depende del estado y la entrada.

3.13 Lenguaje de descripción Verilog

Verilog es un lenguaje para la descripción de sistemas digitales (HDL: *Hardware Description Language)*. Los sistemas pueden ser descritos:

- Nivel estructural empleando elementos de librería o bien elementos previamente creados, se realiza la interconexión de unos con otros. Seria similar a una captura esquemática donde la función del diseñador es instanciar bloques y conectarlos entre si.
- Nivel de comportamiento el diseñador describe la transferencia de información entre registros (nivel RTL: *Register Transfer Level*).

Estos dos niveles de descripción pueden mezclarse, dando lugar a los denominados diseños mixtos. Existen multitud de lenguajes HDL en el mercado (de hecho inicialmente cada fabricante disponía de su propio lenguaje), sin embargo la necesidad de unificación ha hecho que en la actualidad sólo existan dos grandes lenguajes: VHDL y Verilog. Ambos están acogidos a estándares IEEE (VHDL en 1987 y Verilog en 1995). Existen defensores y detractores de cada uno de ellos. Con carácter general se dice que es más fácil aprender Verilog al ser un lenguaje mas compacto. Verilog nació en 1985 como un lenguaje propietario de una compañía (*Cadence Design System*), pero en 1990 se formó OVI (*Open Verilog International*) haciendo dicho lenguaje de dominio publico, permitiendo a otras empresas que pudieran emplear Verilog como lenguaje, con objeto de aumentar la difusión de dicho lenguaje.

Uno de los aspectos que salta a la vista al contemplar un código Verilog es su similitud con el lenguaje C. Una de las mayores diferencias que presenta este lenguaje es que permite modelar sistemas digitales reales, que funcionan de forma paralela a diferencia de la ejecución secuencial, típica de un sistema computacional.

Verilog permite que en un diseño se puedan usar diferentes niveles de descripción de sistemas digitales en un mismo ambiente; las diferentes descripciones pueden ser utilizadas para verificar el diseño y además pueden ser sintetizadas; esto es, ser traducidas a la interconexión de componentes básicos de un dispositivo programable.

Verilog permite el diseño estructural en base a componentes básicos, así como descripciones más abstractas que se enfocan en el comportamiento del sistema. Este comportamiento puede describirse mediante expresiones lógicas y también empleando procedimientos.

Un diseño basado en descripciones funcionales puede resultar lento y de gran costo en el área. Las descripciones en niveles estructurales permiten optimizar los circuitos lógicos para maximizar la velocidad y minimizar el área. La figura 3-3, resume las etapas del diseño digital.

3.14 Descripción Estructural

Nivel Compuertas

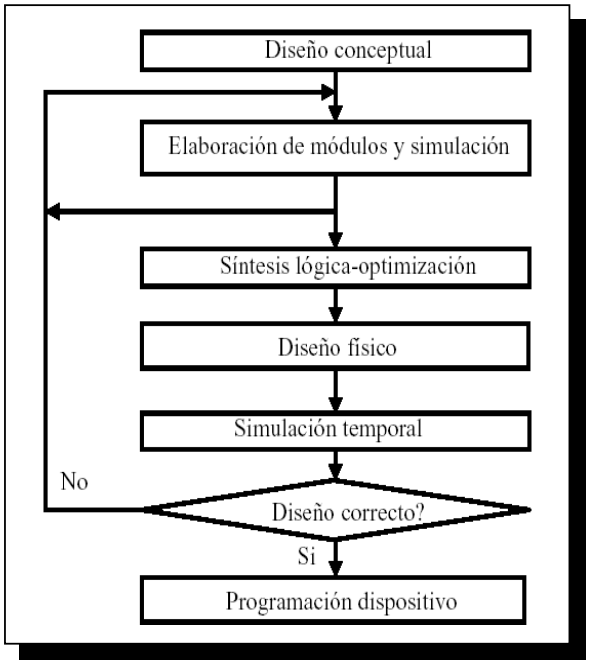


Fig. 3-3 Etapas de diseño digital.

Permite representar una red lógica mediante sus ecuaciones. Para esto, se emplean funciones lógicas básicas para representar compuertas. Los operadores *and, or not,* en minusculas son palabras reservadas, y se emplean como nombres de funciones. Las conexiones o *nets* permiten ir uniendo componentes de un diseño digital.

Procesos

El concepto de procesos que se ejecutan en paralelo es una de las características fundamentales del lenguaje, siendo ese unos de los aspectos diferenciales con respecto al lenguaje procedural como es el C.

El lenguaje Verilog tiene una doble funcionalidad:

1. La realización de simulaciónes de sistemas digitales: empleando dicho lenguaje para describir los estímulos que se van a aplicar, la interconexión entre los bloques.
2. La descripción de un sistema para su posterior síntesis.

Tener un diseño en Verilog que simule correctamente es "relativamente fácil", existiendo múltiples alternativas para conseguirlo. Sin embargo si lo que se pretende es realizar un diseño que no sólo se comporte como está previsto, sino que además sea sintetizable, deben de respetarse ciertas normas, algunas de las cuales pasamos a enumerar:

1. No emplear retrasos, dado que ello conduciría a diseños poco o imposiblemente portables de una tecnología a otra. Los sintetizadores suelen ignorar estos retrasos, pudiendo haber diferencias notables entre la simulación y la síntesis.

2. No modificar una misma variable en dos procesos diferentes. En este caso la modificación de la misma variable en dos procesos diferentes puede dar lugar a resultados diferentes de la simulación y de la síntesis (en caso de que se pueda sintetizar).

3. En una asignación procedural, todas las variables de las que dependa la asignación deben de aparecer en la lista de sensibilidad. Como la lista de sensibilidad sólo contiene a *sel*, el sintetizador suele suponer que se desea muestrear el valor de a y b cuando se activa sel, por lo que añade biestables controlados por el nivel de sel que muestrean la salida de los multiplexores.

4. Si no se define completamente una variable, el lenguaje supone que conserva el último valor asignado, sintetizándose un biestable que almacene su estado.

5. Los sintetizadores no suelen admitir procesos *initial*.

6. Los sintetizadores no suelen admitir los operadores división y resto.

7. Debe tenerse precaución con el problema de carrera (*race condition*), que se produce cuando dos asignaciones se realizan en el mismo instante pero una depende de la otra, y por tanto el orden de ejecución es importante.

3.15 Acerca del lenguaje

Verilog es un lenguaje de descripción hardware (*Hardware Description Language, HDL*) utilizado para describir sistemas digitales, tales como procesadores, memorias o un simple flip-flop. Esto significa que realmente un lenguaje de descripción hardware puede utilizarse para describir cualquier hardware (digital) a cualquier nivel.

La descripción del sistema puede ser tan sencilla como la de un flip-flop, tal y como se refleja en la Figura 3-4, o un sistema complejo de más de un millón de transistores, tal es el caso de un procesador. Verilog es uno de los estándares HDL disponibles hoy en día en la industria para el diseño hardware. Este lenguaje nos permite la descripción del diseño a diferentes niveles, denominados niveles de abstracción.

3.16 Niveles de abstracción en Verilog

Verilog soporta el diseño de un circuito a diferentes niveles de abstracción, entre los que destacan:

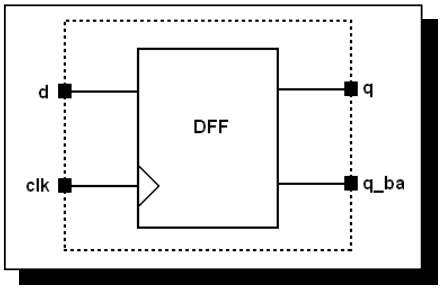


Fig. 3-4 Descripción de un flip – flop.

- Nivel de puerta. Corresponde a una descripción a *bajo nivel* del diseño, también denominada modelo estructural. El diseñador describe el diseño mediante el uso de primitivas lógicas (AND, OR, NOT, etc...), conexiones lógicas y añadiendo las propiedades de tiempo de las diferentes primitivas. Todas las señales son discretas, pudiendo tomar únicamente los valores '0', '1', 'X' o 'Z' (siendo 'X' estado indefinido y 'Z' estado de alta impedancia).

La figura 3-5, representa la descripción de un multiplexor a nivel de puertas.

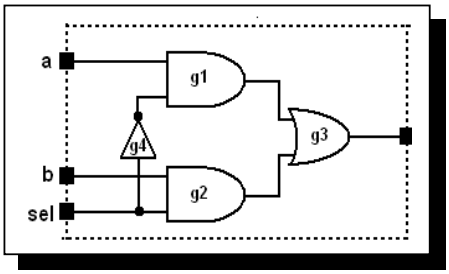


Fig. 3-5 Multiplexor a nivel de compuertas.

- Nivel de transferencia de registro o nivel RTL. Los diseños descritos a nivel RTL especifican las características de un circuito mediante operaciones y la transferencia de datos entre registros. Mediante el uso de especificaciones de tiempo las operaciones se realizan en instantes determinados. La especificación de un diseño a nivel RTL le confiere la propiedad de diseño sintetizable, por lo que hoy en día una moderna definición de diseño a nivel RTL es todo código sintetizable se denomina código *RTL*. Le corresponde a la descripción a nivel RTL de un *flip-flop*. Este nivel de descripción, por la propiedad de ser sintetizable, será el nivel utilizado por excelencia en el diseño HDL.

- Nivel de comportamiento *(Behavioral level*) 1. La principal característica de este nivel es su total independencia de la estructura del diseño. El diseñador, más que definir la estructura, define el comportamiento del diseño. En este nivel, el diseño se define mediante algoritmos en paralelo. Cada uno de estos algoritmos consiste en un conjunto de instrucciones que se ejecutan de forma secuencial.

- La descripción a este nivel pude hacer uso de sentencias o estructuras no sintetizables, y su uso se justifica en la realización de los denominados *testbenches*.

La descripción de un diseño en Verilog comienza con la sentencia:

module <nombre_módulo> <(definición las señales de interfaz)>;

En segundo lugar se declaran las entradas/salidas:

input/output <width> señal;

Seguidamente se describe el módulo/diseño; y se termina la descripción con:

endmodule //Notar que a diferencia de las demás sentencias, no se introduce ";"
La figura 3-6, muestra las diferentes partes en la descripción de un diseño.

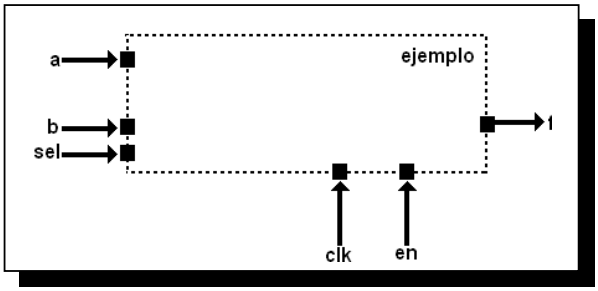


Fig. 3-6 Descripción de un diseño.

3.17Algunas consideraciones acerca del lenguaje

Antes de proseguir con la definición del diseño es importante hacer notar las siguientes consideraciones.

- Comentarios. De una sola línea: pueden introducirse precedidos de //. De varias líneas: pueden introducirse con el formato */* comentario */*.
- Uso de Mayúsculas. Verilog es sensible al uso de mayúsculas. Se recomienda el uso únicamente de minúsculas.
- Identificadores. Los identificadores en Verilog deben comenzar con un carácter, pueden contener cualquier letra de la a a la z, caracteres numéricos además de los símbolos "_" y "$". El tamaño máximo es de 1024 caracteres.

3.18 Números en Verilog

Las constantes numéricas en Verilog pueden especificarse en decimal, hexadecimal, octal o binario. Los números negativos se representan en complemento a 2 y el carácter "_" puede utilizarse para una representación más clara del número, si bien no se interpreta.

La sintaxis para la representación de una constante numérica es:

<Tamaño>'<base><valor>

Si bien el lenguaje permite el uso de números esteros y reales, por brevedad y simplicidad sólo se utilizara la representación de constantes numéricas enteras.

Entero	Almacenado como	Descripción
1	00000000000000000000000000000001	Unzised 32 bits
8'hAA	10101010	Sized hex
6'b10_0011	100011	Sized binary
'hF	00000000000000000000000000001111	Unzised hex 32 bits
6'hCA	001010	Valor truncado
6'hA	001010	Relleno de 0's a la izquierda
16'bz	zzzzzzzzzzzzzzzz	Relleno de z's a la izquierda
8'bx	xxxxxxxx	Relleno de x's a la izquierda
8'b1	00000001	Relleno de 0's a la izquierda

Fig. 3-7 Definición de números enteros.

La figura 3-7, muestra algunos ejemplos de definición de números enteros. Verilog expande el valor hasta rellenar el tamaño especificado. El número se expande de derecha a izquierda siguiendo los siguientes criterios:

- Cuando el tamaño es menor que el valor se truncan los bits más significativos.
- Cuando el tamaño es mayor que el valor, se rellenan con el valor del bit más significativo del número, siguiendo la forma que se muestra en la . Figura 2-3.

Bit más significativo	**Se rellena con**
0	0
1	0
z	z
x	x

Fig. 3-8 Convención de tamaño de valor.

Los **números negativos** se especifican poniendo el signo "-" delante del tamaño de la constante, tal y como se especifica a continuación:

-8'd2 11111110.

La representación interna de los números negativos en Verilog se realiza en complemento a 2.

3.19 Tipos de datos

Existen en Verilog dos tipos de datos principalmente:

- Nets. Representan conexiones estructurales entre componentes. No tienen capacidad de almacenamiento de información. De los diferentes tipos de *nets* sólo utilizaremos el tipo *wire*.

- Registers. Representan variables con capacidad de almacenar información. De los diferentes tipos de *registers* sólo son utilizados el tipo *reg* y el tipo *integer* (estos últimos solo en la construcción de los testbenches).

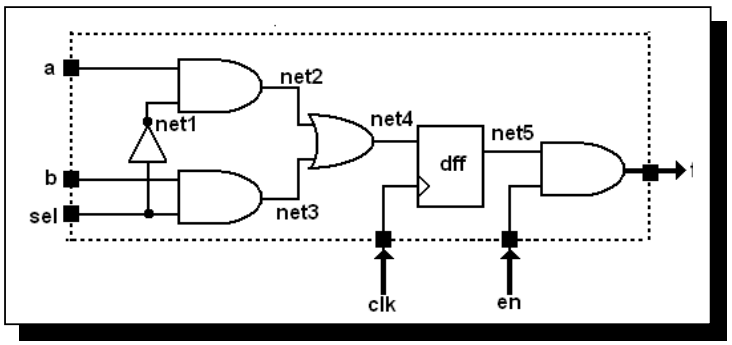


Fig. 3-9 Descripción con nodos internos.

Las señales de interfaz y los nodos internos se declaran de la siguiente forma:

- Inputs. El tipo de las señales de entrada NO SE DEFINEN, por defecto se toman como wire.
- Outputs. Las salidas pueden ser tipo *wire* o *reg*, dependiendo si tienen capacidad de almacenamiento de información. OJO, en Verilog, un nodo tipo *wire* puede atacar a una salida.
- Nodos internos. Siguen la misma filosofía que las salidas.

No piense el usuario que la declaración de un nodo tipo *reg* lleva asociado la síntesis del mismo mediante un elemento secuencial, tal y como se observa en la Figura 3-9. La Figura 3-10 muestra la descripción del ejemplo1 declarando net4 tipo reg. Si observamos el diseño final vemos que es idéntico. La diferencia está en la forma de definir el código.

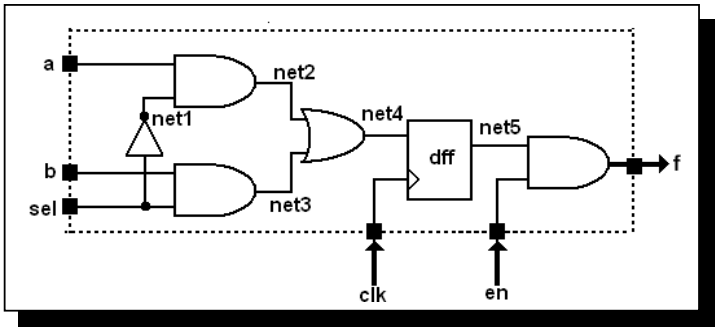


Fig. 3-10 Descripción con nodos internos con net4 tipo reg.

3.20 Procesos

El concepto de procesos que se ejecutan en paralelo es una de las características fundamentales del lenguaje, siendo ese uno de los aspectos diferenciales con respecto al lenguaje procedural como el lenguaje C.

Toda descripción de comportamiento en lenguaje Verilog debe declararse dentro de un proceso, aunque existe una excepción que trataremos a lo largo de este apartado. Existen en Verilog dos tipos de procesos, también denominados *bloques concurrentes*.

Initial. Este tipo de proceso se ejecuta una sola vez comenzando su ejecución en tiempo cero. Este proceso NO ES SINTETIZABLE, es decir no se puede utilizar en una descripción RTL. Su uso está íntimamente ligado a la realización del *testbench*.

Always. Este tipo de proceso se ejecuta continuamente a modo de bucle. Tal y como su nombre indica, se ejecuta siempre. Este proceso es totalmente sintetizable. La ejecución de este proceso está controlada por una temporización (es decir, se ejecuta cada determinado tiempo) o por eventos. En este último caso, si el bloque se ejecuta por más de un evento, al conjunto de eventos se denomina *lista sensible*. La sintaxis de este proceso es pues:

always <temporización> o <@(lista sensible)>

La sentencia muestra dos ejemplos de utilización de los procesos *initial* y *always*.

initial	always @(a or b or sel)
begin	begin //Sobra en este caso
clk = 0;	if (sel == 1)
reset = 0;	y = a;
enable = 0;	else
data = 0;	y = b;
end	end //Sobra en este caso

De estos ejemplos se pueden apuntar las siguientes anotaciones:

- *Begin, end*. Si el proceso engloba más de una asignación procedural (=) o más de una estructura de control (*if-else*, *case*, *for*, etc…), estas deben estar contenidas en un bloque delimitado por *begin* y *end*.

- Initial. Se ejecuta a partir del instante cero y, en el ejemplo, en tiempo 0 (no hay elementos de retardo ni eventos, ya los trataremos), si bien las asignaciones contenidas entre *begin* y *end* se ejecutan de forma secuencial comenzando por la primera. En caso de existir varios bloques *initial* todos ellos se ejecutan de forma concurrente a partir del instante inicial.

- Always. En el ejemplo, se ejecuta cada vez que se produzcan los eventos variación de la variable *a* o variación de *b* o variación de *sel* (estos tres eventos conforman su lista de sensibilidad) y en tiempo 0. En el ejemplo, el proceso *always* sólo contiene una estructura de control por lo que los delimitadores *begin* y *end* pueden suprimirse.

- Todas las asignaciones que se realizan dentro de un proceso *initial* o *always* se deben de realizar sobre variables tipo *reg* y NUNCA sobre nodos tipo *wire*. La Figura 3-12 muestra un ejemplo de asignación errónea sobre nodos tipo *wire* en un proceso *initial*.

En general, la asignación dentro de un proceso *initial* o *always* tiene la siguiente sintaxis:

variable = f(wire,reg,constante numérica)

El siguiente código muestra un ejemplo de asignación errónea sobre nodos tipo *wire* en un proceso *initial*.

```
wire clk,reset;
reg enable,data;
initial
begin
clk = 0; //Error
reset = 0; //Error
enable = 0;
data = 0;
end
```

```
reg clk,reset;
reg enable,data;
initial
begin
clk = 0;
reset = 0;
enable = 0;
data = 0;
end
```

La variable puede ser tanto una variable interna como una señal del interfaz del módulo. La asignación puede ser de un nodo tipo *wire*, de una variable tipo *reg*, de una constante numérica o una función de todas ellas.

A pesar de que las asignaciones, procedurales o continuas, se ejecutan secuencialmente, es posible controlar el instante en que se producen. Esto se consigue mediante el uso de temporizaciones o eventos.

3.21 Eventos de flanco

Este evento se produce por la combinación de flanco/s de subida y/o bajada de una variable simple o de una lista sensible.

Evento	**Descripción**
always @(posedge clk or posedge mr) b <= b+c;	Cada vez que se produce un flanco de subida de clk o de mr se evalúa la expresión
always @(posedge clk or negedge mr) b <= b+c;	Cada vez que se produce un flanco de subida de clk o de bajada de mr se evalúa la expresión.

4

Software: ISE, ACTIVE - HDL

4.1 Software ISE Project Navigator de Xilinx.

Actualmente cualquier proceso de ingeniería dispone de un soporte software que asiste al ingeniero de aplicaciones o sistemas en el desarrollo de sistemas complejos. Los sistemas electrónicos reconfigurables del tipo FPGA son un buen ejemplo de la complejidad que se puede alcanzar, esta complejidad no sería abarcable sin la ayuda de un entorno con herramientas que asistan en el proceso de diseño, simulación, síntesis del resultado y configuración del hardware. Un ejemplo de un entorno de este tipo es el software de la empresa Xilinx denominado ISE (*Integrated Software Environment*).

Este software constituye un verdadero entorno EDA (*Electronic Desing Automation*). La Figura 4-1, representa el esquema de los componentes más importantes del ISE y la secuencia en que se utilizan. La interfaz gráfica de usuario (GUI: *Graphic User Interface*) se denomina *Project Navigator* y facilita el acceso a todos los componentes del proyecto. Los diseños de usuario se pueden introducir mediante diferentes formatos. Los más utilizados son: los esquemáticos, los grafos de estados y las descripciones hardware en VHDL. Una vez compilados los diseños se puede simular su comportamiento a nivel funcional o a nivel temporal. A nivel funcional no tiene en cuenta los retardos provocados por el hardware y a nivel temporal simula el diseño teniendo en cuenta cómo se va a configurar el hardware.

4.2 Entorno del Project Navigator

El *Project Navigator* es el entorno de diseño que permite programar las tarjetas de desarrollo de la familia Xilinx. El *project navigator* es un manejador de alto nivel que permite diseñar en dispositivos CPLD o FPGA, permitiendo realizar cualquiera de las siguientes funciones:

- Crear o añadir archivos fuente, los cuales aparecen en la ventana fuentes (*sources*).
- Modificar el archivo fuente en el espacio de trabajo.
- Arrancar un proceso en los archivos fuente desde la ventana de procesos.
- Ver la salida de un proceso en la ventana de estado.

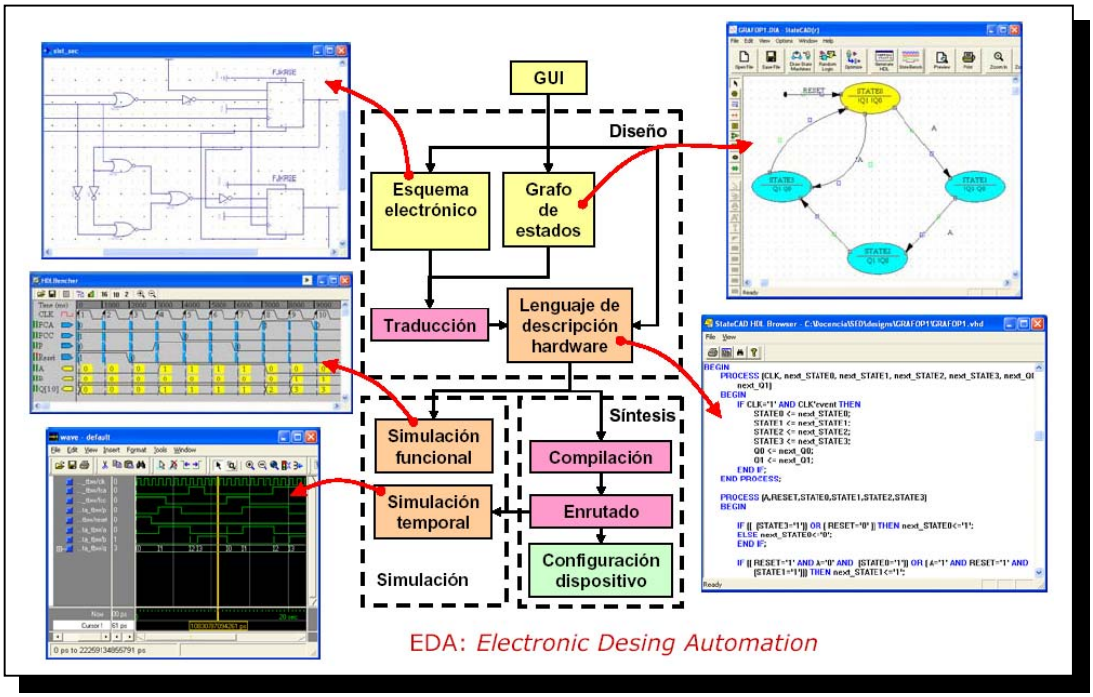


Fig. 4-1 Pasos en el desarrollo de aplicaciones con FPGA.

La figura 4-2, muestra la ventana del entorno de desarrollo del *Project Navigator*.

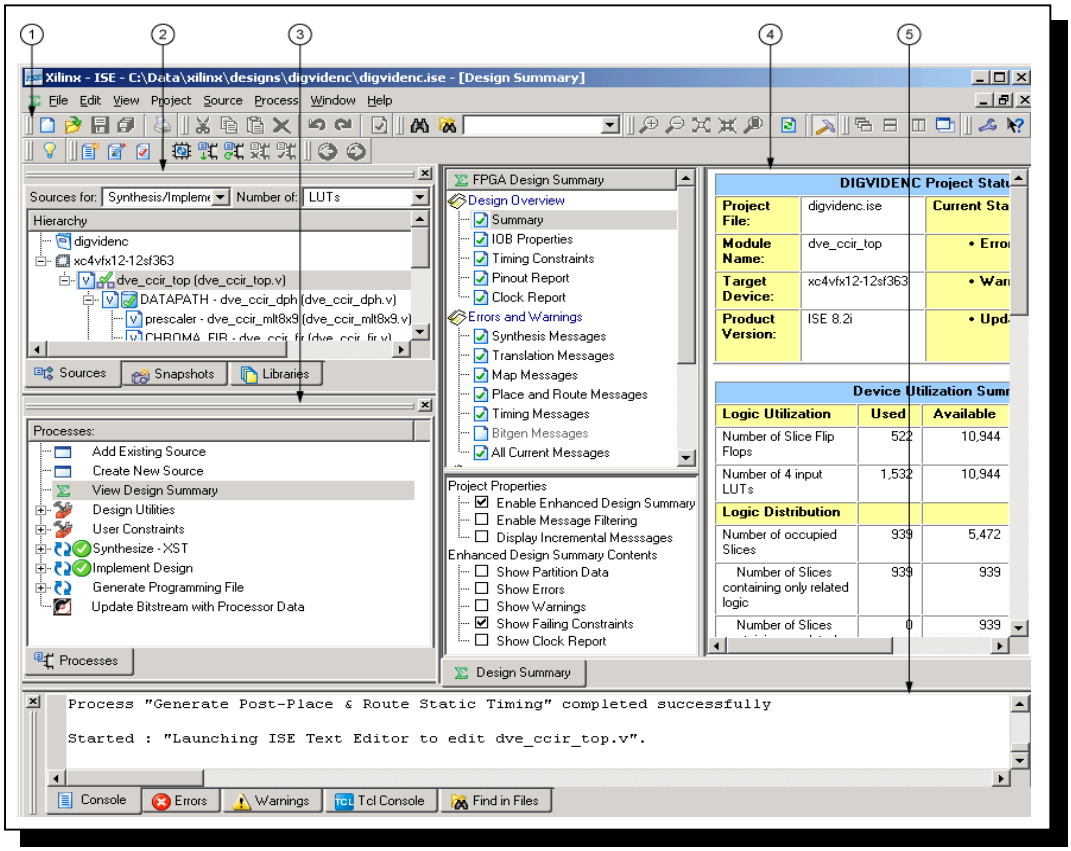


Fig. 4-2 Ventana principal del Project Navigator.

Descripción:

1. Barra de Herramientas
2. Ventana fuente
3. Ventana de procesos
4. Espacio de trabajo
5. Ventana de mensajes.

4.3 Foundation ISE (*Integrated System Environment*).

Esta herramienta es valida para el diseño de sistemas digitales, con las familias de CPDLs y FPGAs que comercializa Xilinx.

Se dispone de una versión gratuita denominada *Webpack.* Esta versión esta limitada en cuanto a la complejidad del sistema que se puede implementar: http://www.xilinx.com.

Jerarquía:

Una vez que se ha creado el proyecto, la parte de la pantalla de Jerarquía, muestra diversos tipos de archivos que son:

- Nombre del proyecto (*Project Name*). El nombre se determina una vez que el proyecto ha sido creado.

- Documentos de usuario (*User Documents*). Cualquier archivo incluido en el proyecto que no es parte del diseño.

- Dispositivo para el proyecto (*Project device Entry*). Define el tipo de dispositivo, empaque, velocidad, Herramienta de síntesis y simulación para el diseño, esta información es ingresada cuando se crea el proyecto y puede ser modificada en cualquier momento.

- Archivos fuente (*Sources files*). Definen parte del diseño

- Modulo principal (*Top Module*). Define que modulo es el principal en la jerarquía del diseño con propósito de implementación. La implementación del proceso de diseño comienza siempre en el modulo que ha sido designado como Top. Cualquier modulo de la pantalla de jerarquía puede ser designado como modulo Top.

- Particiones (*partitions*). Define un modulo fuente que ha sido marcado para reutilizarse, las particiones pueden ser archivos HDL, esquemáticos, o módulos fuente EDIF de cualquier nivel de jerarquía.

Vista de diseño

En la ventana fuentes (*sources*) se pueden ver y editar proyectos y archivos fuente de acuerdo a diferentes vistas de diseño, seleccionando una vista de diseño de la ventana "*Sources for*", mostrando una lista que brinda las opciones:

- Síntesis/Implementación. En esta vista de diseño, el proyecto y todos los archivos fuente del diseño que están asociados a la síntesis / implementación serán mostrados, en esta vista, se pueden ingresar diseños, síntesis, implementación y configuración del dispositivo a programar.

- Vista de simulación. Distintas vistas de simulación están disponibles, cada una representa una vista del diseño para simulación a una fase distinta del flujo. Por ejemplo, en la vista de de simulación comportamental se pueden tener archivos HDL, pruebas de banco, y/o formas de onda de pruebas de banco.

Es posible cambiar las propiedades del proyecto que se este desarrollando, tales como la familia de la tarjeta a utilizar, tipo de modulo de alto nivel, la herramienta de síntesis, el simulador, y el lenguaje de simulación generado.

Dependiendo del archivo fuente, así como de la herramienta que se elija para trabajar, pestañas adicionales pueden estar disponibles en la ventana de "*Sources*":

- Siempre disponibles: *Souces tab, Snapshots tab, Libraries tab*.
- Editor de restricciones: *Timing Constraints tab*.
- Editor de "*floorplan"*.

Tipos de procesos

Los siguientes tipos de procesos están disponibles cuando se trabaja en el diseño:

- Tareas: Cuando se arranca un proceso de tarea, el software ISE inicia un modo lote, esto es, el software procesa el archivo fuente, pero no abre ninguna herramienta de software adicional en el espacio de trabajo. La salida del proceso aparece en la ventana de estado.

- Reportes: La mayoría de las tareas incluyen reportes de sub-procesos, los cuales generan un reporte de estatus, por ejemplo, un reporte de síntesis, o bien de mapeo, cuando se genera un proceso de reporte, este aparece en el espacio de trabajo.

- Herramientas: Cuando se lleva a cabo un proceso de herramientas, la herramienta relacionada aparece de modo independiente en el espacio de trabajo, donde se puede ver o modificar los archivos fuente del diseño.

4.4 Flujo de diseño básico para FPGA

Cuando se realiza un diseño para un FPGA de un nivel de complejidad bajo o moderado, se pueden seguir los siguientes pasos:

1. Crear un proyecto en ISE como sigue:
   (a) Crear un proyecto
   (b) Crear archivos y añadirlos al proyecto, incluyendo el archivo de restricciones de usuario (UCF).
   (c) Añadir los archivos existentes al proyecto.
   (d) Editar los archivos de diseño para especificar su funcionalidad.

(e) Opcionalmente, utilice las plantillas del lenguaje para ayudar en el diseño.
(f) Editar los bancos de prueba del diseño o los archivos de formas de onda que manejan los estímulos para la prueba de archivos de diseño de manera opcional.
(g) Asignar restricciones, de tiempo, asignación de pines, y área de restricciones.

2. Arranque la simulación comportamental (también conocida como simulación RTL).

3. Repetir los pasos 1 y 2 hasta que la funcionalidad deseada es obtenida.

4. Sintetizar el diseño.

5. Implementar el diseño como sigue:
   (a) Iniciar el proceso de Implementar el diseño en el modulo Top, el cual arranca automáticamente el siguiente proceso:
   - Traducir (*Translate*)
   - Mapear (*Map*)
   - Colocar ruta (*Place and Route*)

   (b) Revisar los reportes generados por el proceso de implementar diseño, así como el reporte de Mapeo y el de Colocar ruta, y se cambia una de las siguientes características para mejorar el diseño:
   - Propiedades del proceso
   - Restricciones
   - Archivos fuente

   (c) Modificar el diseño como sea necesario, simular, sintetizar, e implementar el diseño nuevamente hasta que los requerimientos del diseño sean cumplidos.

6. Inicie la simulación de tiempos para verificar la funcionalidad final.

7. Programe su dispositivo Xilinx como sigue:
   (a) Crear un archivo de programación (BIT) para programar el FPGA
   (b) Generar un archivo PROM, ACE o JTAG para depurar o descargar al dispositivo.
   (c) Programar el dispositivo a través de un cable de programación.

4.5 Inicio del software ISE (*Integrated Software Environment*) de Xilinx.

Se comienza ejecutando el software gestor de proyectos mediante el comando:

Inicio>Programas>Xilinx ISE 9.1i>Project Navigator

A continuación se abre la ventana principal como muestra la Figura 4-3.

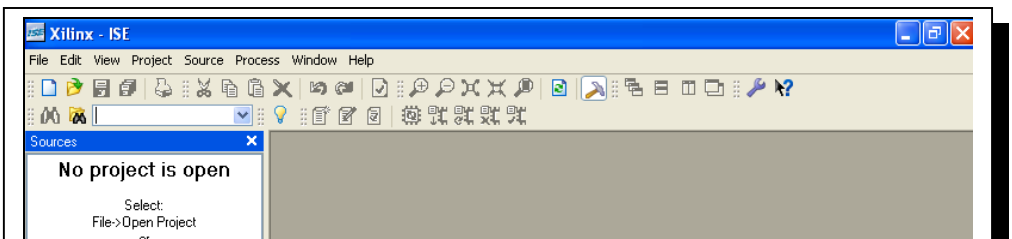


Figura 4-3. Ventana principal de Project Navigator.

4.6 Creando un Nuevo Proyecto en ISE

Un proyecto es una colección con todos los archivos necesarios para crear y descargar un diseño a cualquier dispositivo FPGA o CPLD de Xilinx.

Para crear un nuevo proyecto se deben seguir los siguientes pasos:

1. Seleccionar *File* > *New Project*. Aparecerá el asistente de nuevos proyectos.

2. Primero, seleccionamos la localización (o *path*) del directorio en el que se ubicará el proyecto.

3. Dar clic en *nombre del proyecto* en el campo Project Name.

Cuando se teclea el *nombre_proyecto* en este campo, un subdirectorio del mismo nombre se creará automáticamente en la dirección que se selecciono previamente.

4. Seleccionar *schematic* en el campo *Top-Level* Module *Type*, indicando que el archivo jerárquicamente superior y que engloba los demás será un esquemático, en vez de HDL o EDIF.

5. Ingresar el nombre del nuevo proyecto (figura 4-4(a) y (b)).

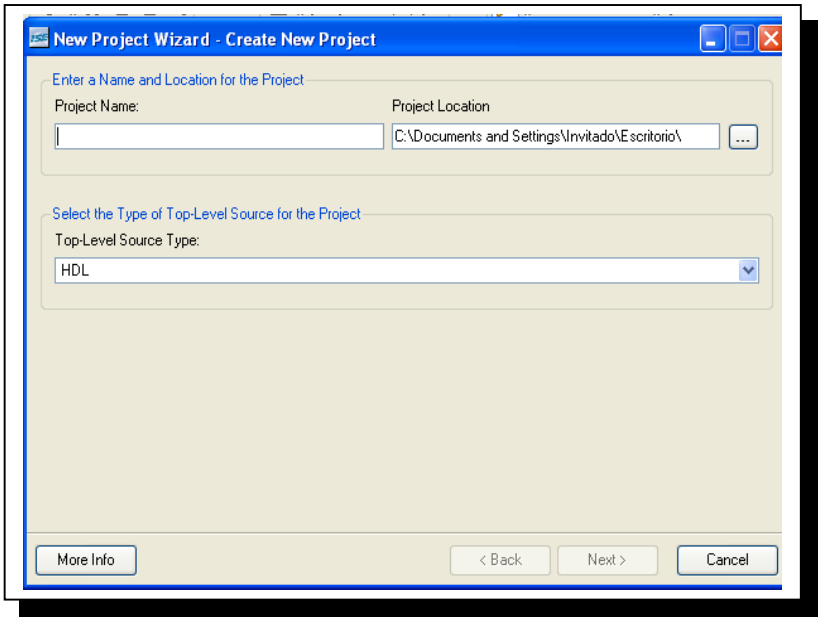
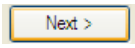


Fig. 4-4a Nombre del proyecto.

6. Dar clic en  para avanzar al siguiente paso.

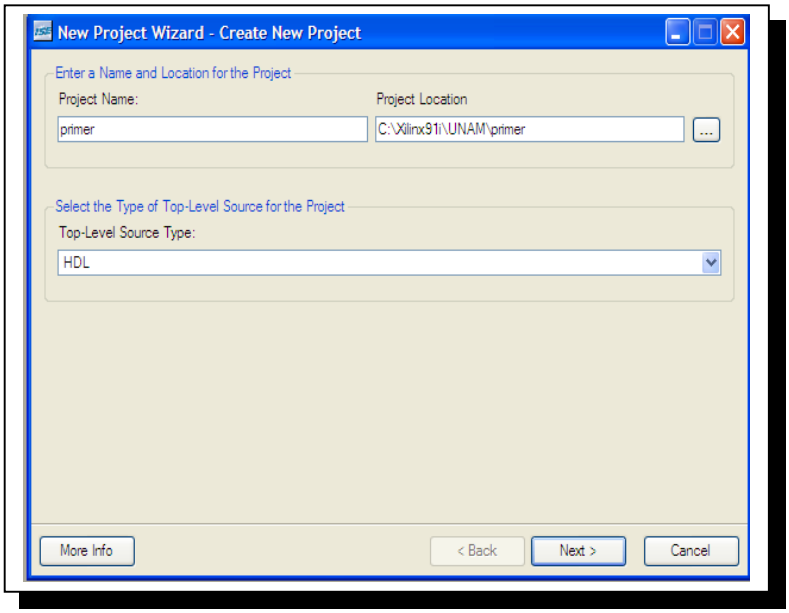
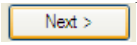


Fig. 4-4b Nombre del proyecto.

4.7 Descripción de las propiedades del dispositivo.

El siguiente paso consiste en llenar los campos que describen el hardware que elegimos para implementar nuestro futuro. Estos contienen las especificaciones del dispositivo de la tarjeta de desarrollo de que se trate (En nuestro caso un FPGA Spartan 3E); a continuación dar clic en  (figura 4-5).

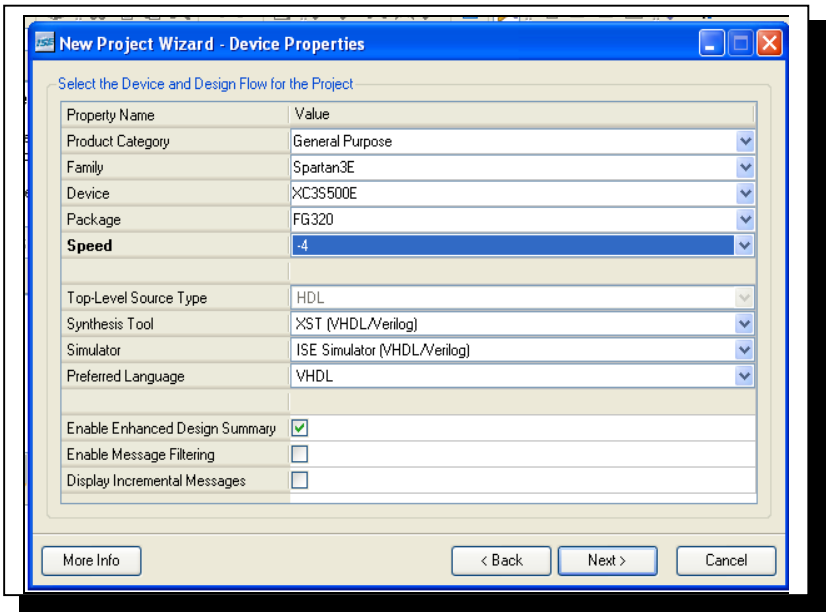
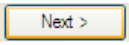


Fig. 4-5 Propiedades del dispositivo FPGA.

A continuación aparece una ventana en la que es posible añadir archivos al proyecto creado, damos clic en 

para obviar los pasos de añadir archivos nuevos o ya creados al proyecto. Esto lo haremos posteriormente (figura 4-6).

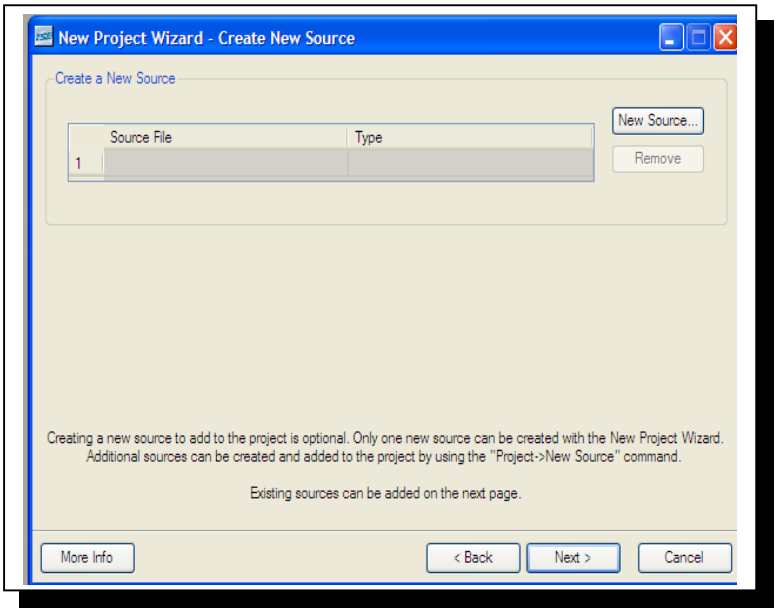
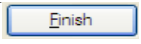


Fig. 4-6 Adición de archivos al proyecto

Las propiedades del proyecto aparecerán resumidas de la siguiente manera en una ventana (figura 4-7), damos clic en 

.

4.8 Creación de un archivo fuente HDL.

Una vez que se crea el proyecto; el siguiente paso consiste en crear un archivo fuente del tipo HDL (VHDL y Verilog).

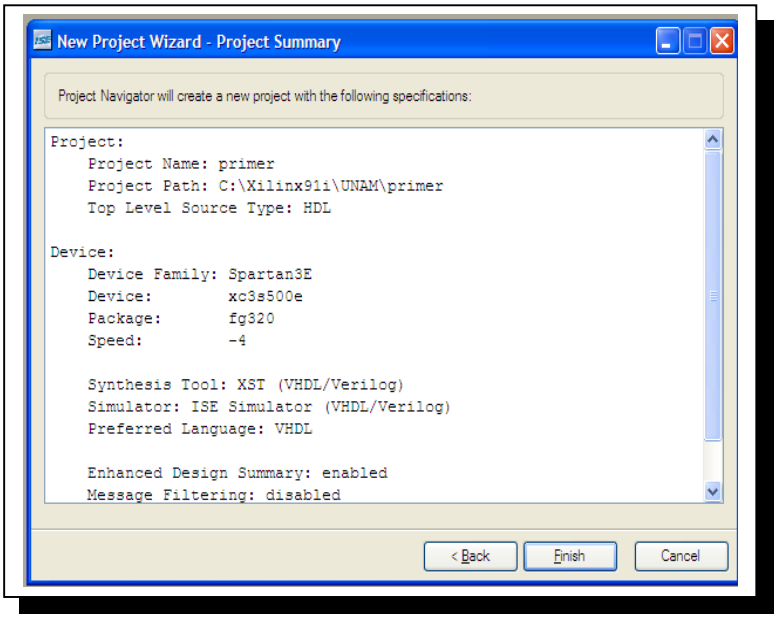


Fig. 4-7 Resumen de las propiedades del proyecto.

En la ventana de proceso es posible elegir entre “Add Existing Source” y “Create New Source” (Añadir un archivo existente o crear uno nuevo), como se observa en la figura 4-8.

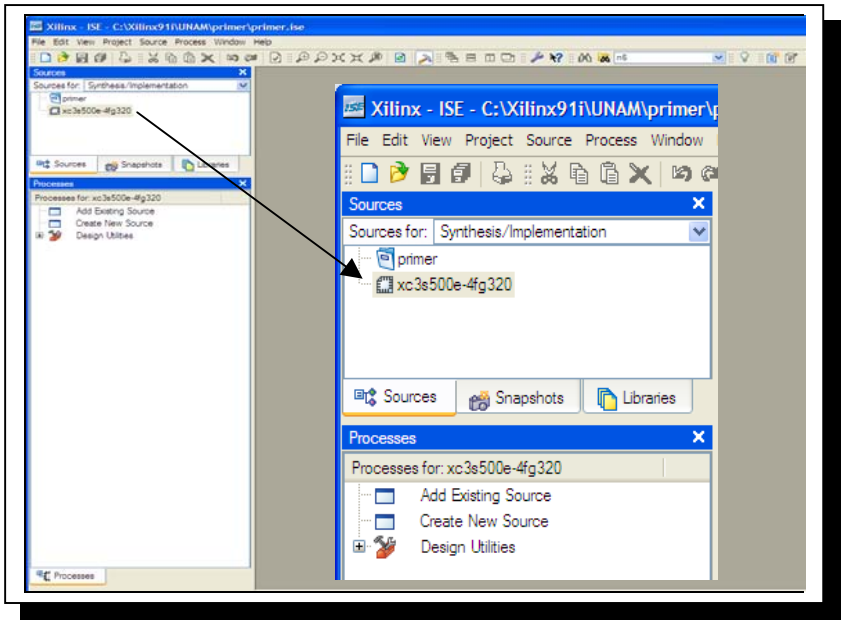


Fig. 4-8 Añadir o crear un archivo fuente.

Con el botón derecho del ratón damos clic en el icono xc3s500e-4fg320 del dispositivo y elegimos “New Source” , para generar un archivo HDL (figura 4-9).

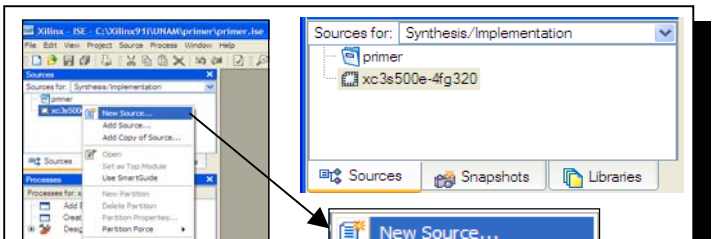


Fig. 4-9 Creación de un archivo fuente.

Elegimos un archivo del tipo "VHDL Module" o bien "Verilog Module", y asignamos un nombre al mismo. Damos clic en Next > (figura 4-10).

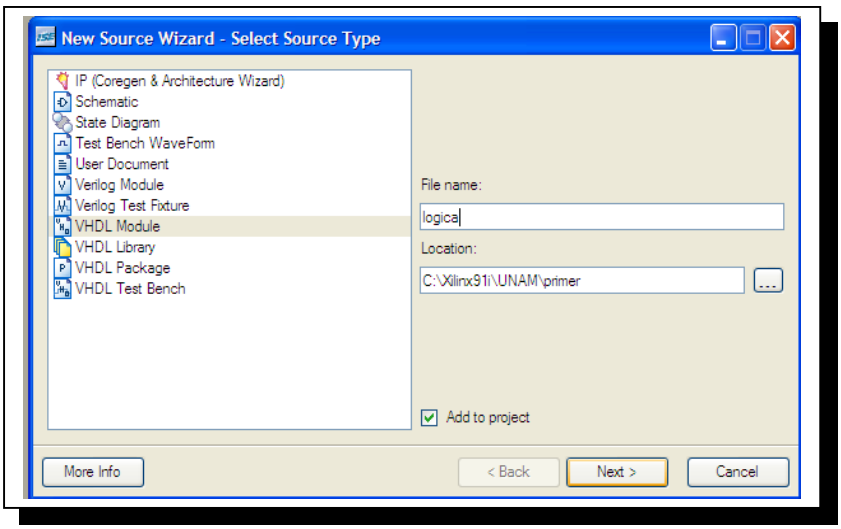


Fig. 4-10 Selección del archivo fuente.

En la siguiente ventana, podemos definir el modulo (cuantas entradas y salidas utilizara nuestro diseño), elegiremos continuar sin definir en este momento el modulo.

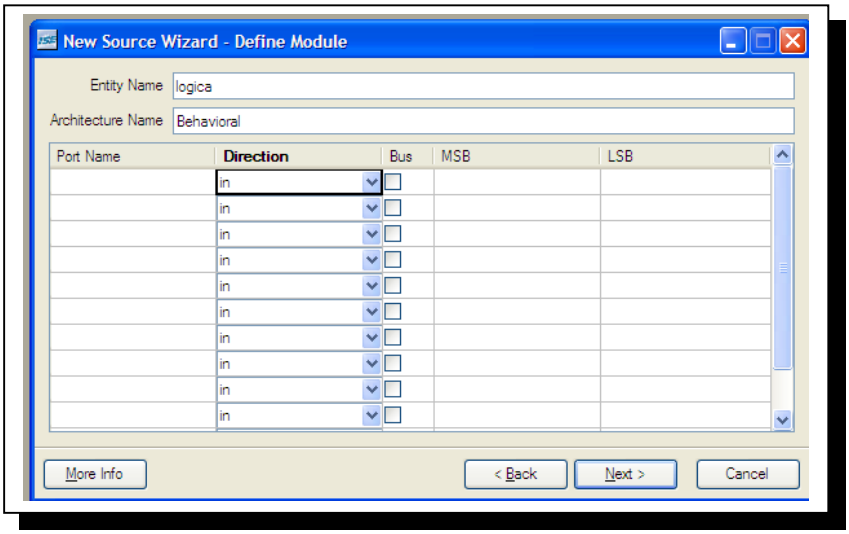


Fig. 4-11 Definir modulo.

En el momento de editar la descripción del circuito lo haremos), dando clic en

Next > , figura 4-11.

La siguinete ventana (figura 4-12), es un resumen de información acerca de archivo fuente creado; damos clic en Finish .

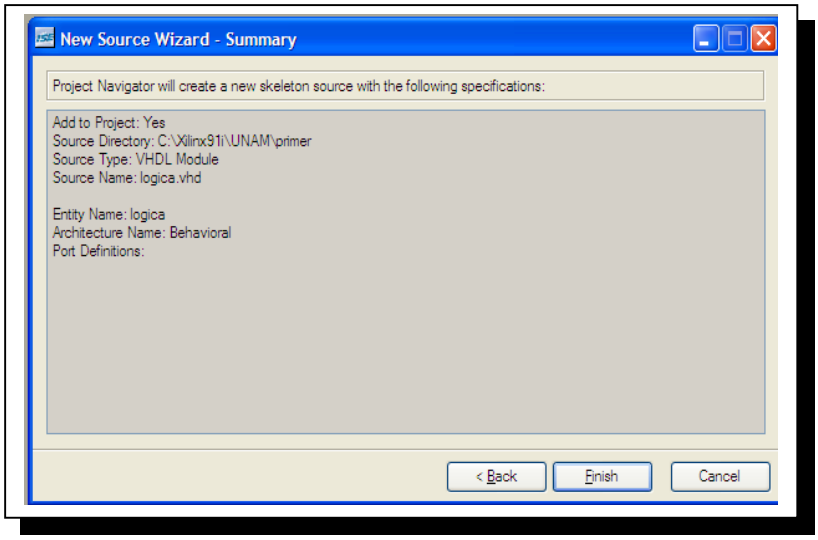


Fig. 4-12 Resumen del archivo fuente.

4.9 Edición del archivo de descripción HDL

En este momento nos encontramos en el entorno que nos permitirá editar en lenguaje de descripción HDL, sea que lo hagamos en VHDl o en verilog, mostrando una pantalla como la que se muestra. Comenzamos, entonces, dando clic en la pestaña lógica.vhd, aparecerá la pantalla de edición e ingresamos el código (figura 4-13).

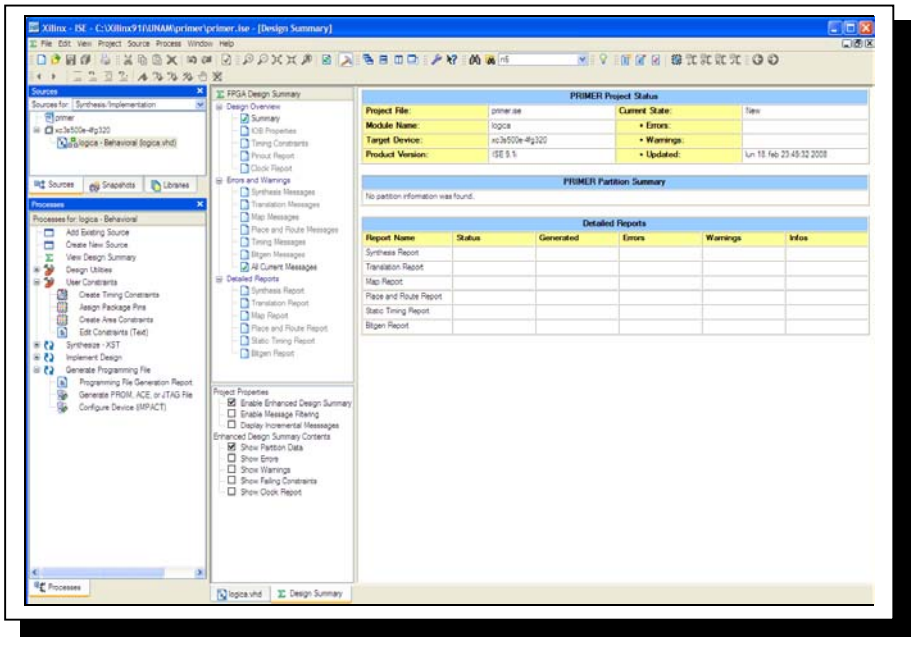
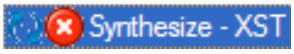


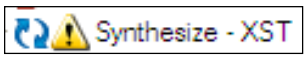
Fig. 4-13 Pantalla de edición

4.10 Síntesis del diseño

Una vez que ya se termino la edición continuamos con el proceso de síntesis; este proceso nos permite comprobar que nuestro código esta sintácticamente correcto (No tiene errores de código). Al término de la ejecución de la síntesis pueden aparecer tres iconos del lado izquierdo del proceso:



Que significa que fallo el proceso de síntesis (Hay uno varios errores en el código)



Que significa que el código tiene una condición anómala; pero que es posible realizar la síntesis, debe verificarse el código.

Significa que la síntesis sucedió satisfactoriamente; es posible continuar con el siguiente proceso.

El proceso de síntesis se arranca dando doble clic en “Synthesize”, y esperar a que termine de ejecutarse la tarea; este proceso termina una vez que aparece la esfera verde con la paloma, como se puede observar en la figura 4-14.

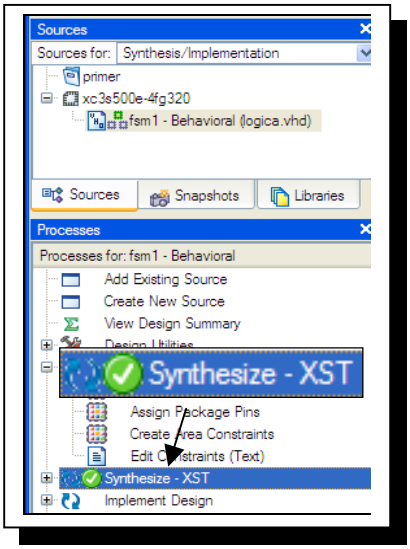


Fig. 4-14 Síntesis correcta.

4.11 Asignación de pines

Después de sintetizar nuestro diseño; se debe efectuar la asignación de pines de entradas y salidas del diseño al correspondiente del dispositivo FPGA de la tarjeta Spartan 3E, para ello debemos ejecutar la línea de proceso “*Assign package pins*” dando doble clic (figura 4-15).

Esta tarea abrirá primero una ventana de dialogo como la que se muestra en la figura 4-16. Esta ventana de dialogo nos avisa acerca de la necesidad de crear un archivo de restricciones (ucf), y si en principio aceptamos que lo cree el Project Navigator. Damos clic en “Yes”.

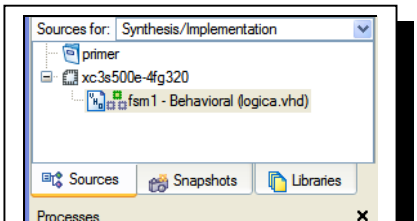


Fig. 4-15 Tarea de asignación de pines.

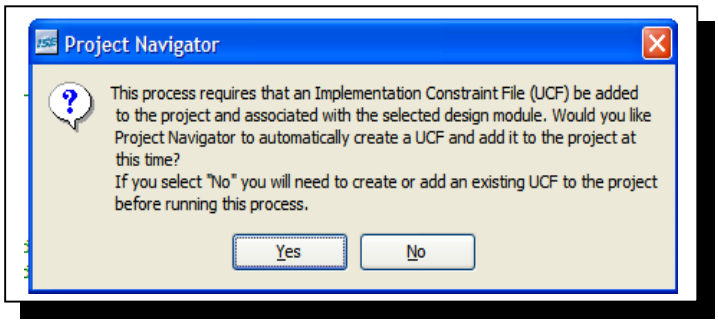


Fig. 4-16 Aviso de creación de un archivo de restricciones.

Se abre ahora otra aplicación del Project Navigator llamada "PACE"; en esta ventana tenemos un área en la que debemos llenar los campos que se nombran "*Loc*" (Ubicación). Debemos referenciarnos con la publicación de Xilinx (*xilinxds312.pdf*) que describe los nombres asignados a todas las localidades de la tarjeta Spartan 3E, esto lo hacemos dando clic en cada uno de los campos en blanco y llenándolos (figura 4-17).

Terminado de asignar los pines, damos clic en el icono guardar , en ese momento podemos cerrar la ventana del "PACE" terminando aquí el proceso de asignación de pines.

4.12 Editar Restricciones ("Constraints")

Procedemos ahora a abrir la opción de editar restricciones dando doble clic en el proceso "*Edit Constraints*". Aparecerá una pantalla como la que se muestra en la figura 4-18.

Se observara que, en efecto se han asignado a cada entrada y salida una localidad de la tarjeta, para agregar ahora las restricciones consultamos nuevamente otra publicación de xilinx (*ug230.pdf)* que nos brinda información acerca de las restricciones ("*constraints*") del hardware de la tarjeta Spartan 3E, con ellas decidimos donde aplicarlas (figura 4-19).

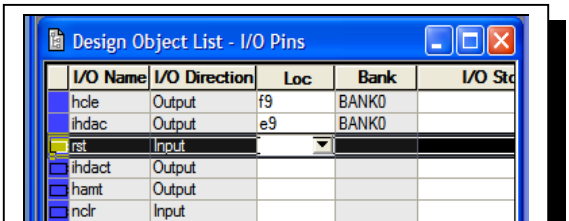


Fig. 4-17 Asignación de pines a entradas y salidas.

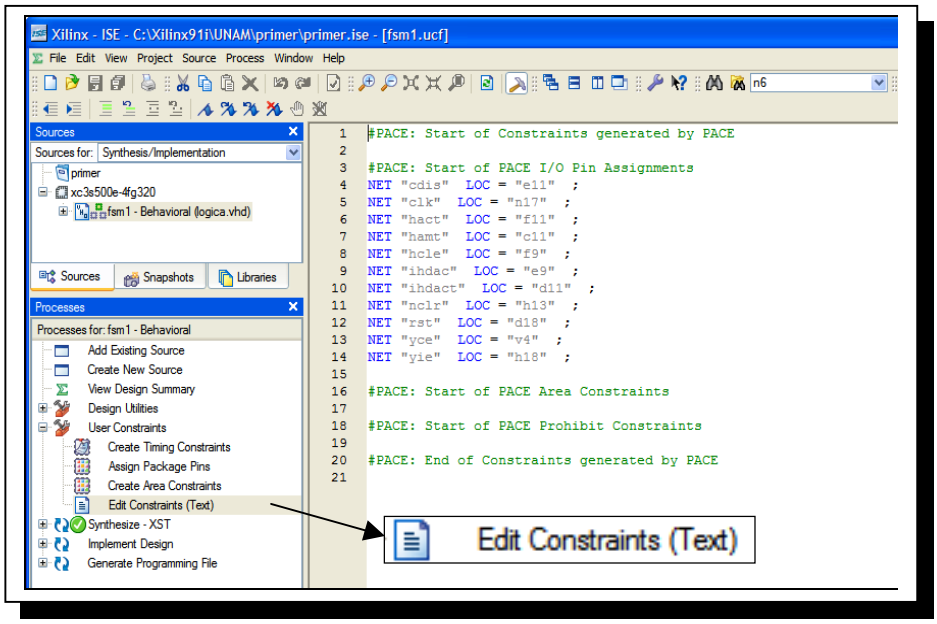


Fig. 4-18 Tarea de edición de Restricciones.

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "cdis"  LOC = "e11"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "clk"  LOC = "n17"  | IOSTANDARD = LVTTL | PULLUP;
NET "hact"  LOC = "f11"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "hamt"  LOC = "c11"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "hcle"  LOC = "f9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "ihdac"  LOC = "e9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8  ;
NET "ihdact"  LOC = "d11"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "nclr"  LOC = "h13" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "rst"  LOC = "d18" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "yce"  LOC = "v4"  | IOSTANDARD = LVTTL | PULLDOWN;
NET "yie"  LOC = "h18"  | IOSTANDARD = LVTTL | PULLUP;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

Fig. 4-19 Edición de restricciones.

Una vez que se han editado las restricciones guardamos nuevamente nuestro archivo de extensión (ucf), dando clic en guardar.

Cerramos esta ventana, en este momento se han asignado los recursos de la tarjeta Spartan 3E a nuestro diseño a implementar.

4.13 Implementar diseño.

El siguiente paso consiste en ejecutar el proceso "*Implement design*". Damos doble clic en la opción de la ventana de proceso con dicho nombre y, esperamos a que termine de ejecutarse esta tarea (figura 4-20), al final de la cual deberá aparecer una pantalla como la que se muestra (Es importante mencionar que cuando la síntesis ocurrió satisfactoriamente, la implementación del diseño ocurre de la misma forma).

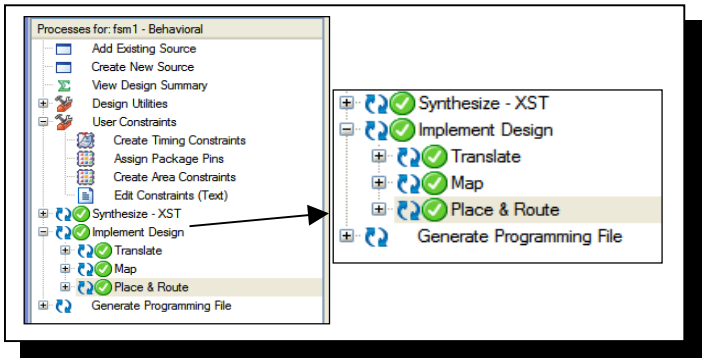


Fig. 4-20 Tarea de Implementar diseño.

Lo anterior nos indicara que la implementación del diseño resulto ser satisfactoria, nos disponemos entonces a continuar con el siguiente paso.

4.14 Generar archivo de programación.

Damos doble clic en la opción de la ventana de proceso "*Generate Programming file*" (Generar el archivo de programación), y esperamos a que se ejecute la tarea (figura 4-21).

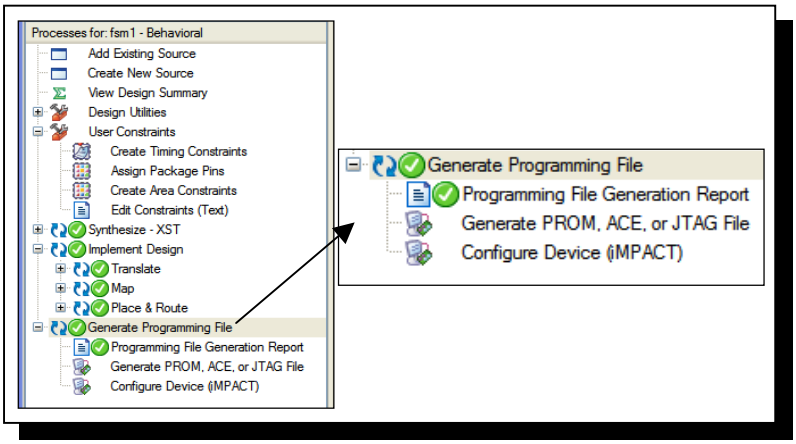


Fig. 4-21 Generar archivo de programación.

Una vez que terminado el proceso, se ha generado el archivo de programación de extensión (bit); que es el archivo que finalmente se descarga al dispositivo FPGA.

4.15 Configurar tarjeta Spartan 3E en Windows

Este procedimiento, aunque en si mismo no forma parte del software ISE Project Navigator; es requerido efectuarse para que la PC pueda detectar la tarjeta Spartan 3E (Nuevo Hardware), para ello requerimos conectar la tarjeta del suministro de alimentación y del puerto de comunicación USB a la PC. Acto seguido colocamos el switch de encendido/apagado en "ON", esto provocara que encienda el led rojo del extremo izquierdo superior de la tarjeta (figura 4-22).

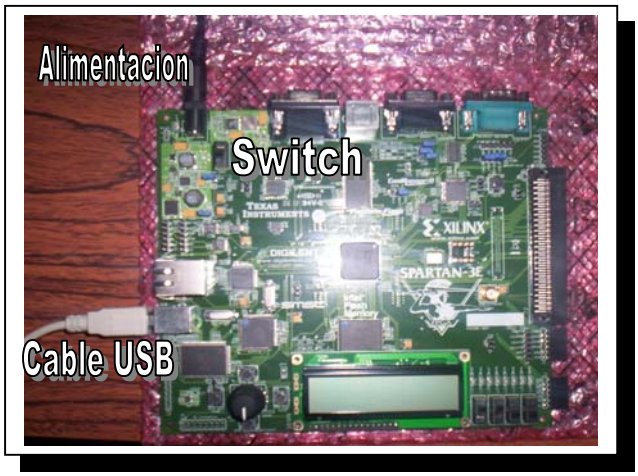


Fig. 4-22 Conexión de la tarjeta Spartan 3E.

La tarjeta Spartan 3E es configurada como cualquier otro dispositivo "*Plug and Play*" (conéctese y úsese), por lo cual al final podemos observarlo en el administrador de dispositivos de Windows (figura 4-23).

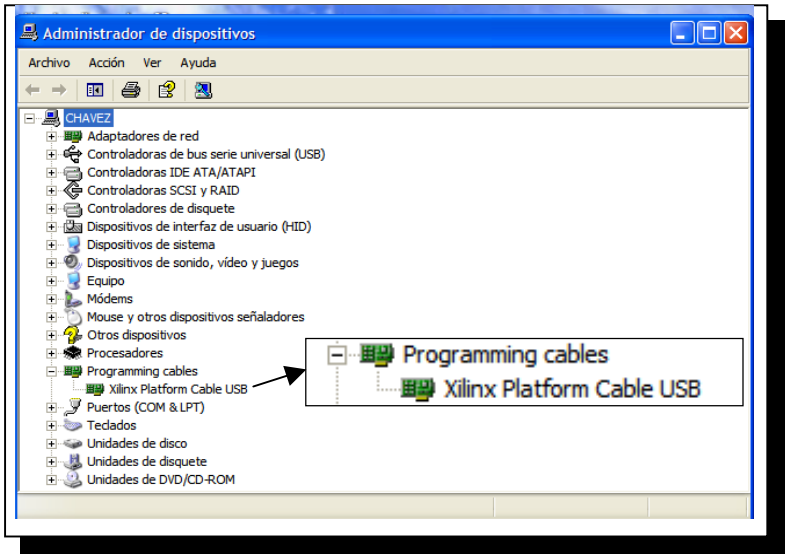
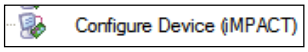


Fig. 4-23 Configuración de tarjeta Spartan 3E.

4.16 Configurar dispositivo

Una vez que nuestra tarjeta esta dada de alta como nuevo hardware, reanudamos el proceso en el entorno ISE Project Navigator; ejecutamos el proceso "*Configure Device*" (Configurar dispositivo) dando doble clic. Lo anterior hace que se muestre una ventana para elegir el tipo de configuración. Dar clic en "*Finish*" con la opción que tiene por defecto "*Automatically connect to…*" (figura 4-24). 

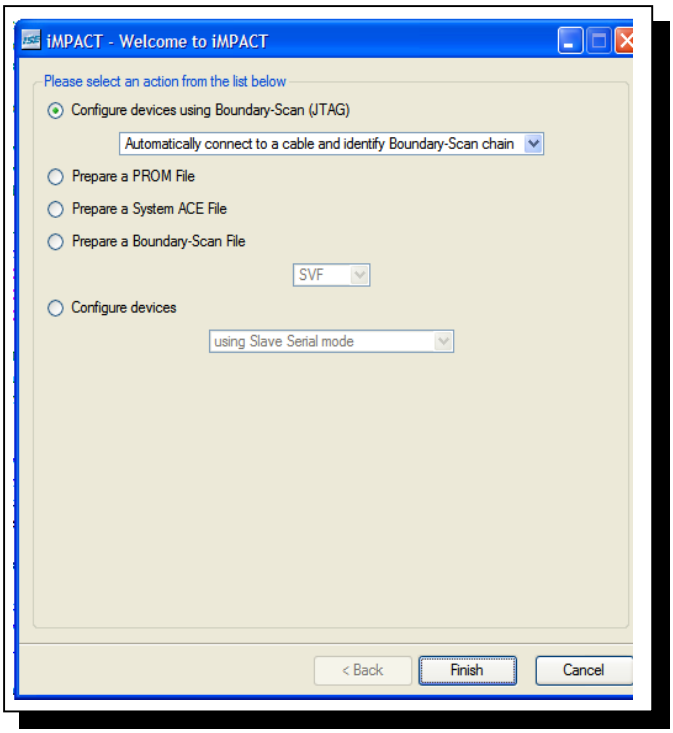


Fig. 4-24 Configuración del dispositivo.

Iniciando la ejecución de esta tarea; aparece un aviso que nos advierte acerca del tiempo que le tomara actualizar el "*firmware*" del cable USB (Cerca de 10 minutos), tiempo en el cual no debemos detener el proceso o desconectar el cable de comunicación (figura 4-25). Damos clic en "Ok".

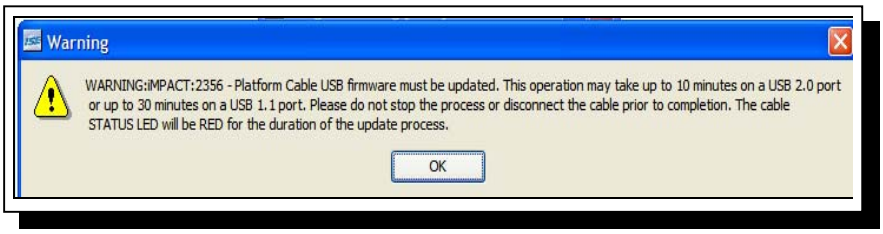


Fig. 4-25 Aviso de descarga del "firmware".

Permitimos que termine de ejecutarse el proceso anterior.

4.17 Programar el dispositivo.

Una vez concluido aparece la siguiente ventana en la que se muestran los dispositivos en los que podemos implementar nuestro diseño, así como una ventana para

elegir el archivo de programación, damos clic en dicho archivo de extensión (bit) y clic nuevamente en "Open" (figura 4-26).

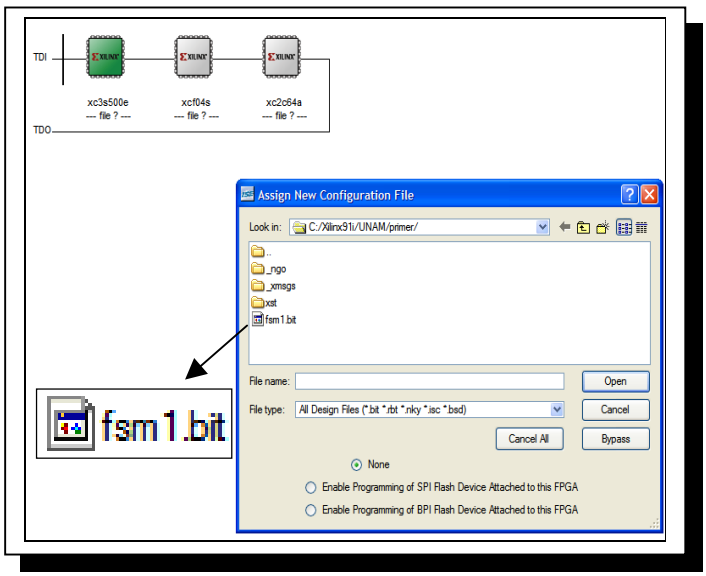


Fig. 4-26 Selección del archivo (bit).

Después de ello, aparecen otras dos ventanas en las daremos clic en "*Bypass*" (en este caso no tenemos archivos con la extensión que solicita en el cuadro de dialogo).

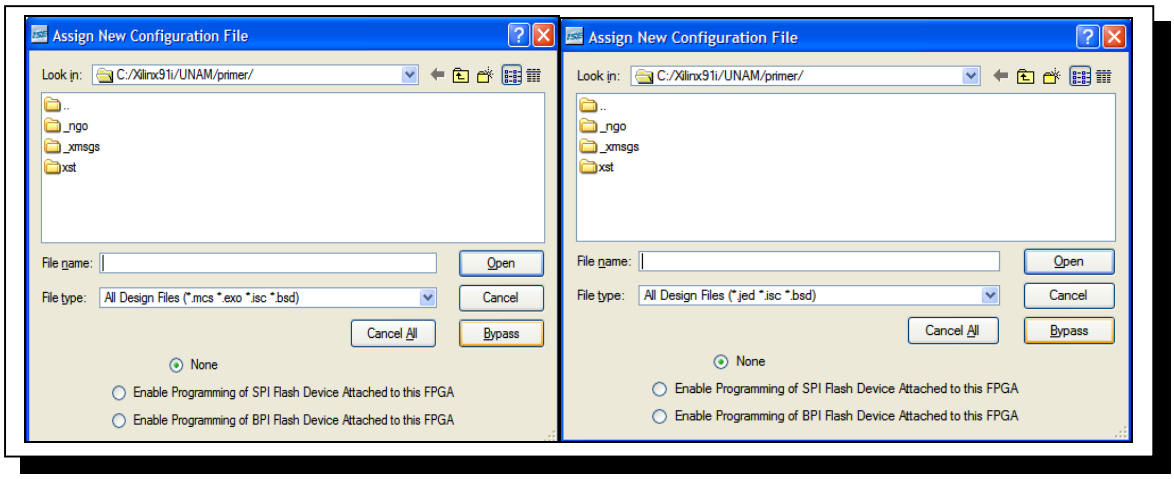


Fig. 4-27 Configuraciones sin utilizar.

El proceso final consiste en programar nuestro dispositivo, damos clic derecho en el icono que contiene el archivo de extensión *.bit, damos otro clic en "Program…" (figura 4-28).

Aparece una ventana de propiedades de programación, en ella verificamos que se encuentre seleccionada la opción "*Pulse Prog*" en las propiedades específicas del dispositivo FPGA, damos clic en "Ok" (figura 4-29).

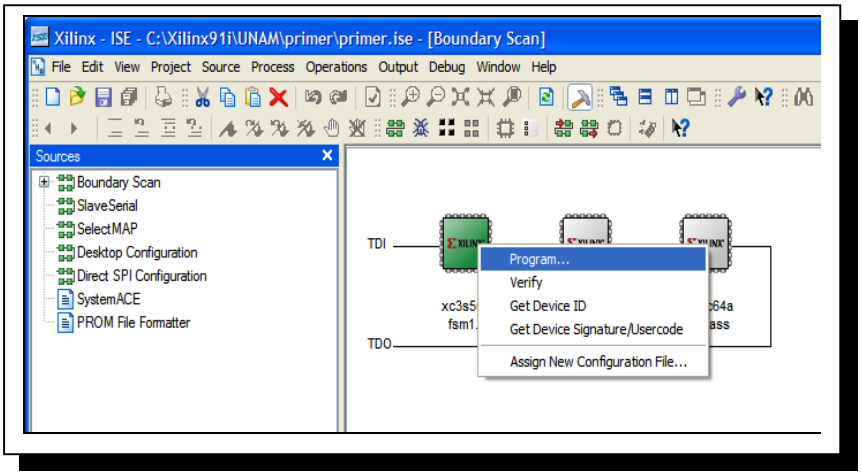


Fig. 4-28 Programación del dispositivo.

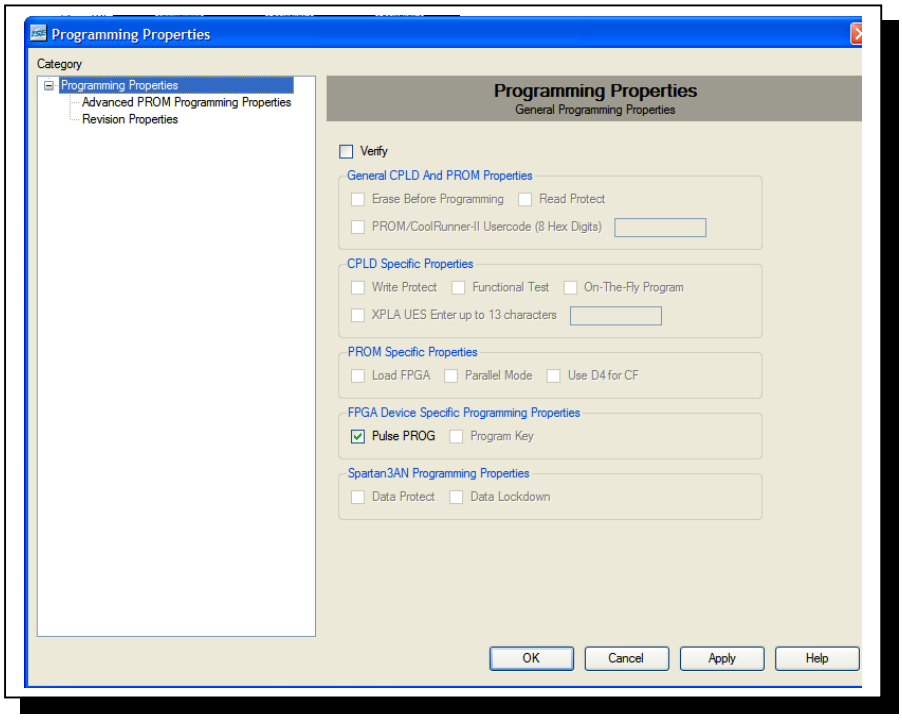


Fig. 4-29 Propiedades de programación.

Se hace la descarga, y se logra programar satisfactoriamente indicado por la leyenda “Program Succeed”, como aparece en la siguiente figura 4-30.

Hasta aquí concluye la síntesis, implementación y programación de la tarjeta Spartan 3E por medio del Software ISE Project Navigator.

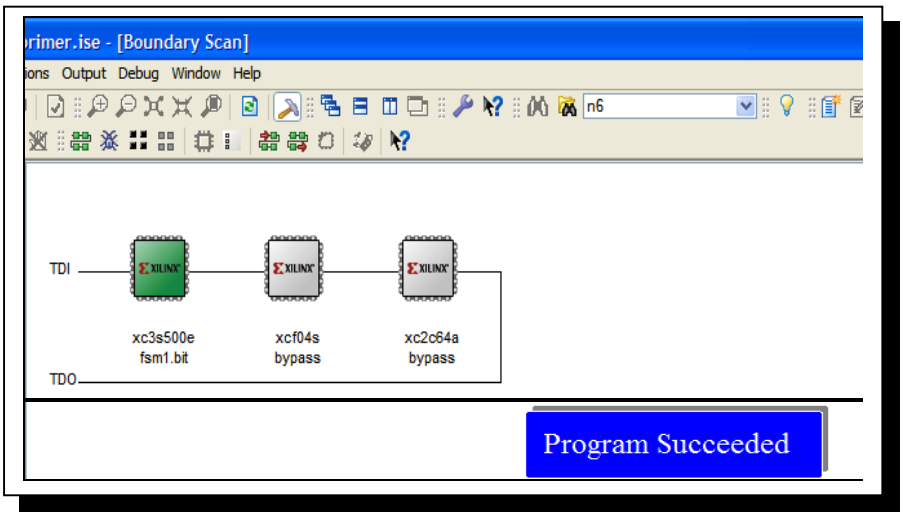


Fig. 4-30 Programación Satisfactoria.

Software Active – Hdl de Aldec.

4.18 Entorno de Active – Hdl para simulación.

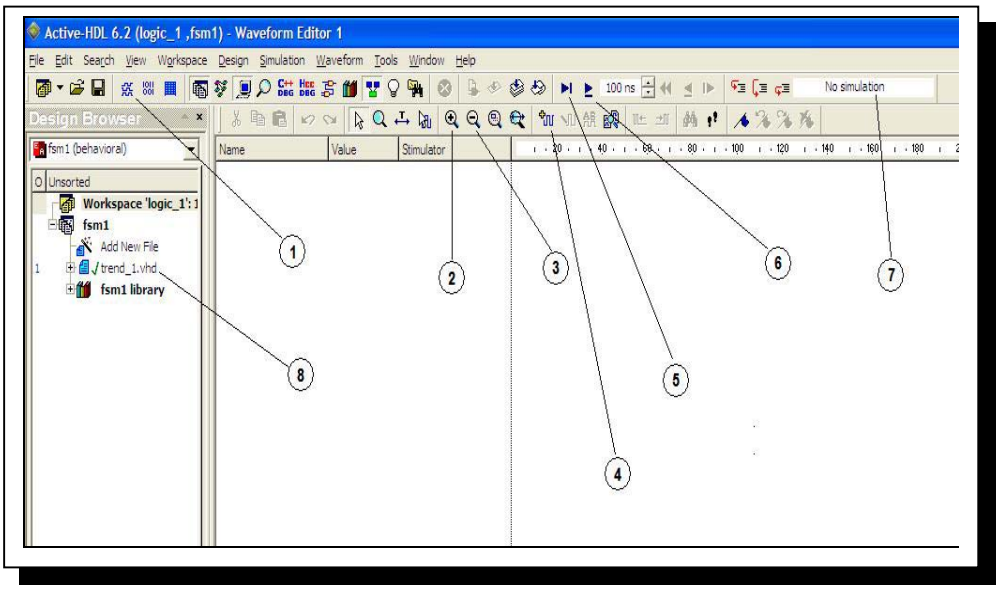


Fig. 4-31 Entorno Active – Hdl para simular.

Descripción de iconos del entorno de simulación.

1. Comenzar nueva simulación
2. Zoom dentro
3. Zoom fuera
4. Agregar señales
5. Simular en el rango..

6. Simular durante
7. Estado de simulación
8. Archivo (y tipo) a simular.

4.19 Iniciar el software Active – Hdl

Para efectuar la simulación de una descripción de un diseño, comenzamos ejecutando el Software Active – Hdl, como sigue.

Dar doble clic en el icono,  para iniciar la ejecución de Active. Una vez que lo ejecute aparecerá una pantalla como se muestra en la figura 4-32.

4.20 Crear un espacio de trabajo

El paso siguiente consiste en crear un espacio de trabajo, el cual nos permitira acceder al menu de trabajo de Active – Hdl. Dar clic en la ruta "File>New> Workspace", para crear un espacio de trabajo (figura 4-33).

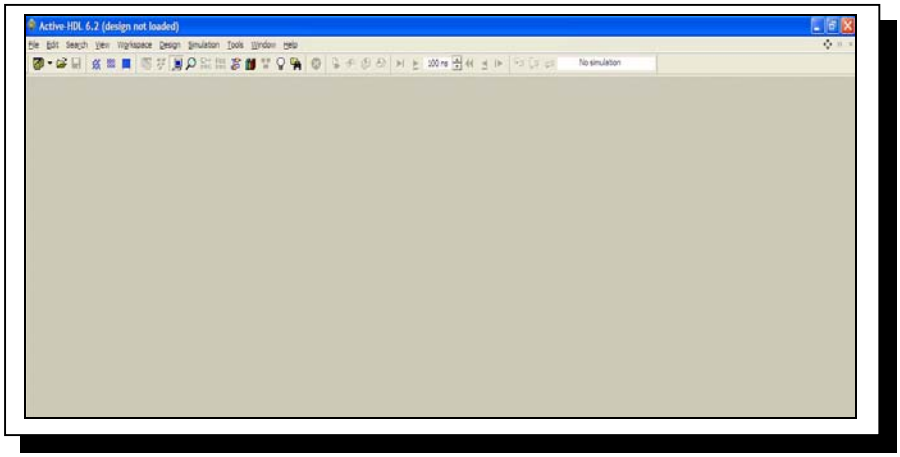


Fig. 4-32 Ventana de incio de Active – Hdl.

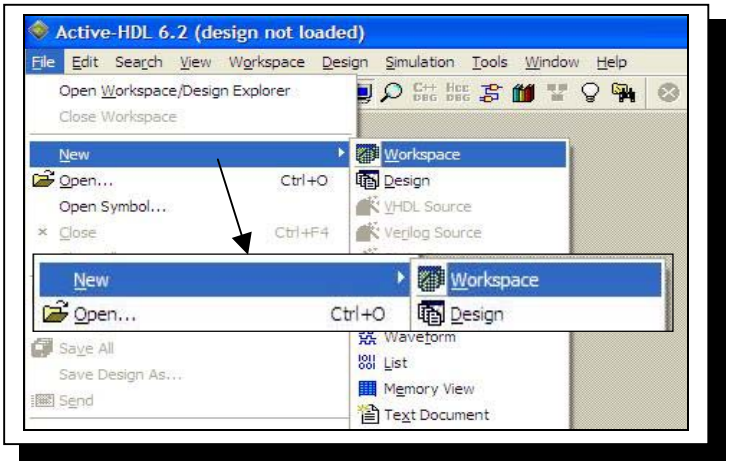


Fig. 4-33 Crear un espacio de trabajo.

Aparece un cuadro de dialogo que solicita ingresar un nombre para dicho espacio de trabajo. Dar clic en OK una vez que se ha ingresado el nombre (figura 4-34).

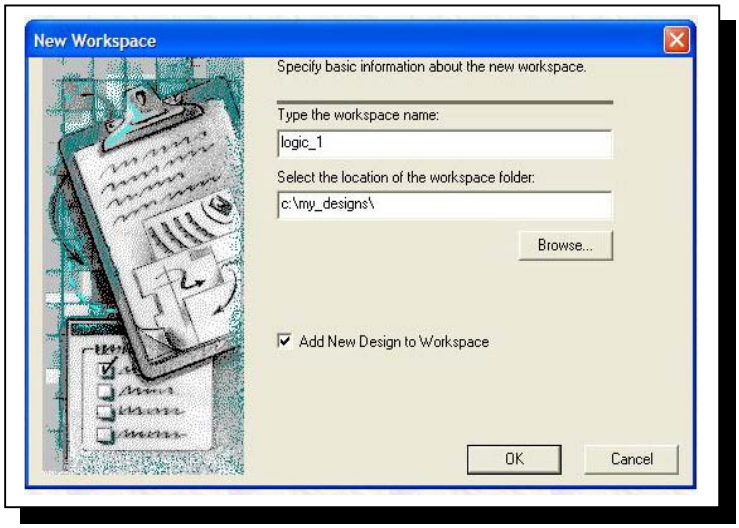


Fig. 4-34 Asignar nombre al espacio de trabajo.

En la siguiente ventana (figura 4-35), nos pide información adicional acerca del nuevo diseño, dejamos los datos que aparecen por defecto. Dar clic en Siguiente > .

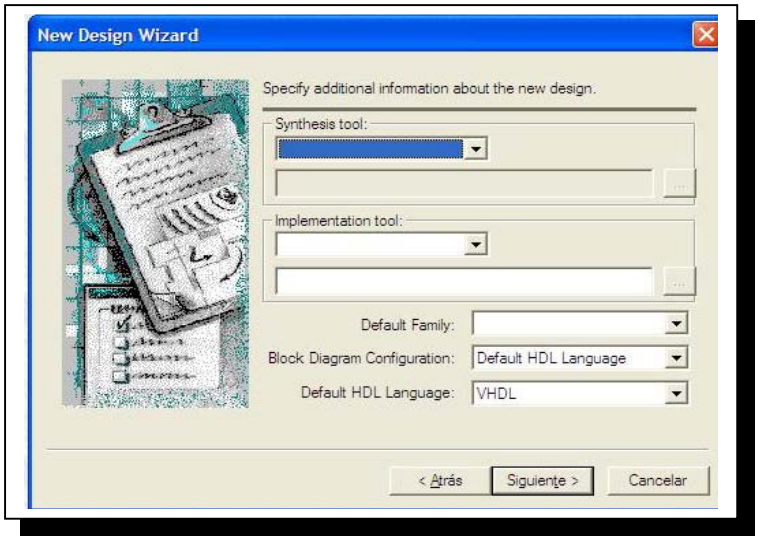


Fig. 4-35 Informacion del diseño.

A continuación nos solicita ingresar un nombre del diseño y el nombre de la librería de trabajo (se deja el mismo nombre). Dar clic en Siguiente > . Ver figura 4-36.

4.21 Crear un modulo HDL

Una vez creado el espacio de trabajo con el nombre del diseño, creamos un modulo hdl, sea en VHDL o en Verilog. Damos doble clic en “Add New File” (figura 4-37).

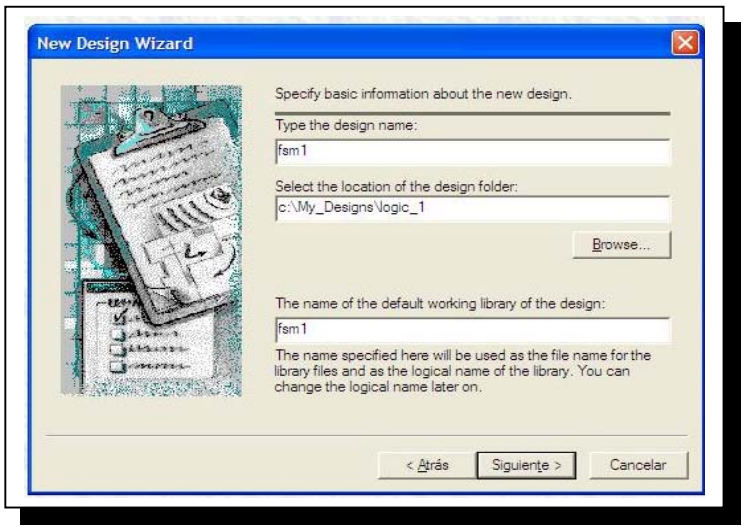


Fig. 4-36 Asignar nombre al diseño.

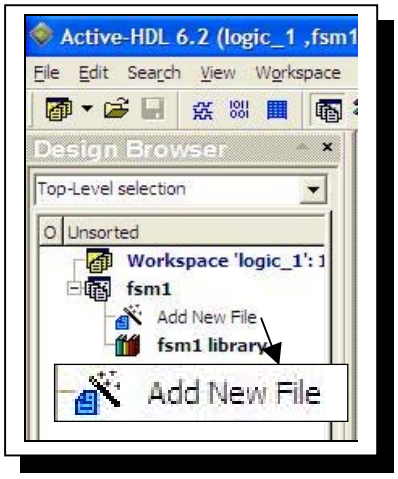


Fig. 4-37 Añadir modulo HDL.

Elegimos “VHDL Source Code” o “Verilog Source Code” dando clic en su icono correspondiente, y posteriormente ingresamos un nombre para este archivo. Hacer clic en aceptar (figura 4-38).

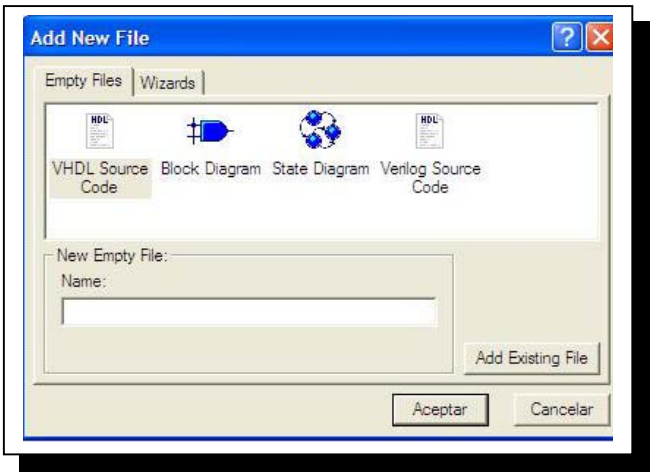


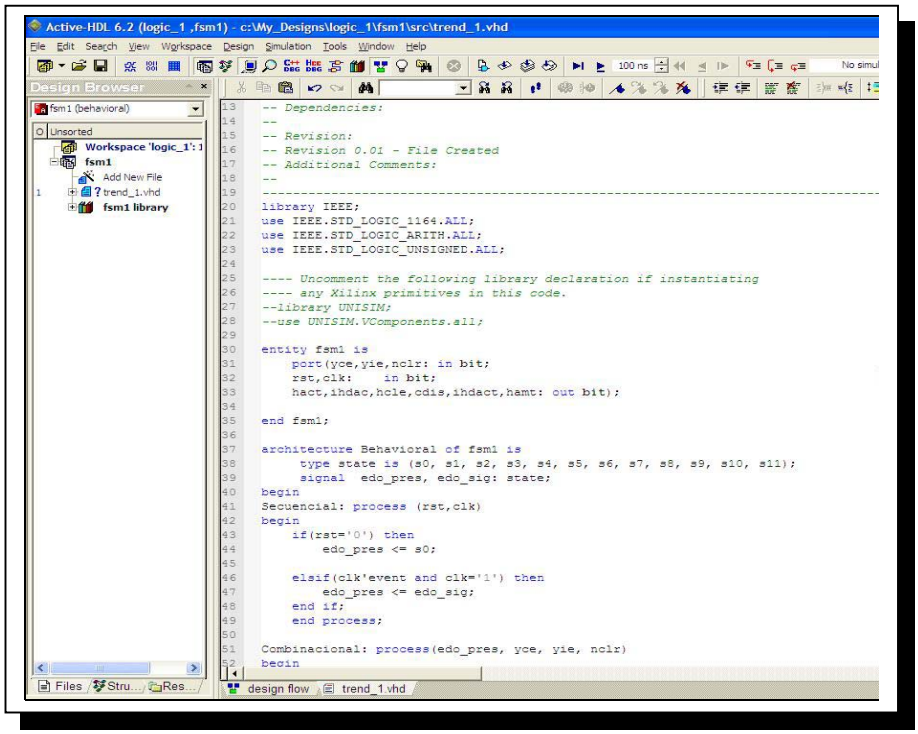
Fig. 4-38 Elección del tipo de archivo HDL.

4.22 Edición del modulo.

Editamos el modulo que recién creamos (sea VHDl o verilog) y lo guardamos una vez que se termine de editar. Asimismo cada que se efectué un cambio debe realizarse la operación de guardar para que se cumplan los cambios efectuados (figura 4-39).

4.23 Compilar el modulo

Compilamos el código depurándolo hasta que no tenga errores. Esto será visible, al aparecer una paloma color verde a la izquierda del modulo. Lo hacemos dando clic en el nombre del archivo con extensión *.vhd o *.v y dando clic en el icono de compilar (figura 4-40).



4-39 Edicion del modulo.

4-40 Compilar el modulo.

4.24 Acceder al editor de simulación

En este punto, nuestro diseño esta listo para efectuar la edición para una simulación temporal. Dar clic en el icono del editor de simulación (figura 4-41).

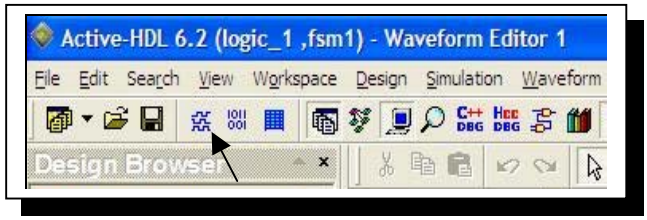


Figura 4-41 Icono del editor de simulación.

La figura 4-42, muestra el entorno de simulación para nuestros diseños. Se observa como se divide por una linea punteada en dos regiones.

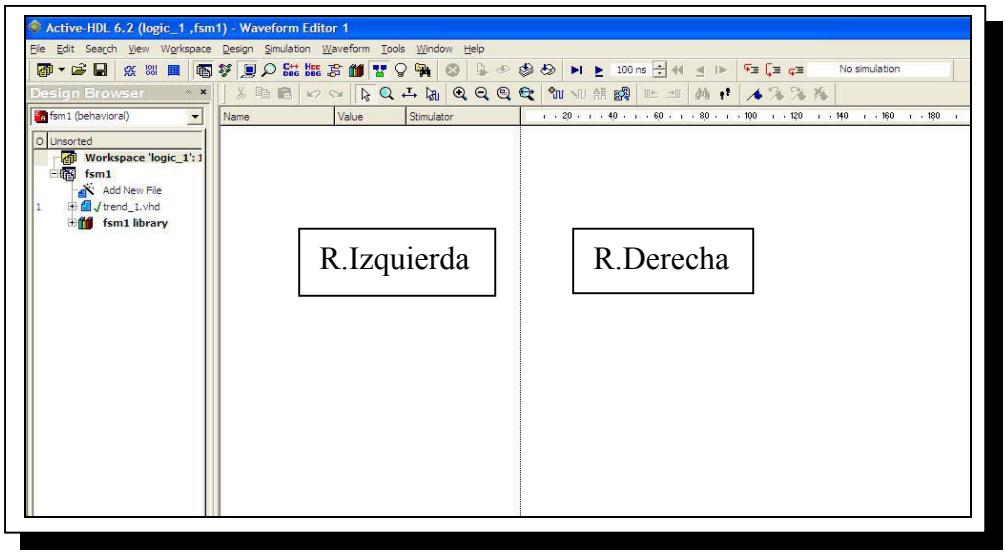


Fig. 4-42 Entorno de edición.

4.25 Agregar las señales

Como pudimos observar en la figura anterior, el editor esta enteramente en blanco. Es necesario agregar las señales que deseamos observar para ser simuladas. Dar clic derecho en la región izquierda, desplegara un menú con la opción "*Add signals*" como única opción habilitada. Dar clic en ella (figura 4-43).

Aparecen en forma de lista las entradas, salidas asi como las señales que estan descritas en el diseño, agregarlas seleccionando y dando clic en "Add" (Se pueden seleccionar todas a la vez utilizando la tecla Shift ⇧, sin soltar dando clic a todas ellas (figura 4-44).

4.26 Estimular las señales

Lo siguiente consiste en agregar estímulos a las señales que aparecen en el editor. Para ejecutar este paso damos clic con el botón derecho del ratón en todas y cada una de las entradas; una a la vez y damos clic en "*Stimulators*" como se observa en la figura 4-45.

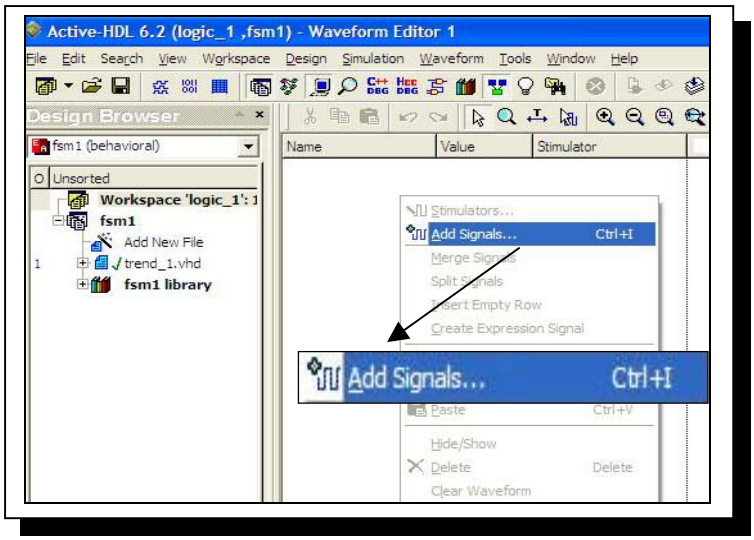


Fig. 4-43 Menu para agregar señales.

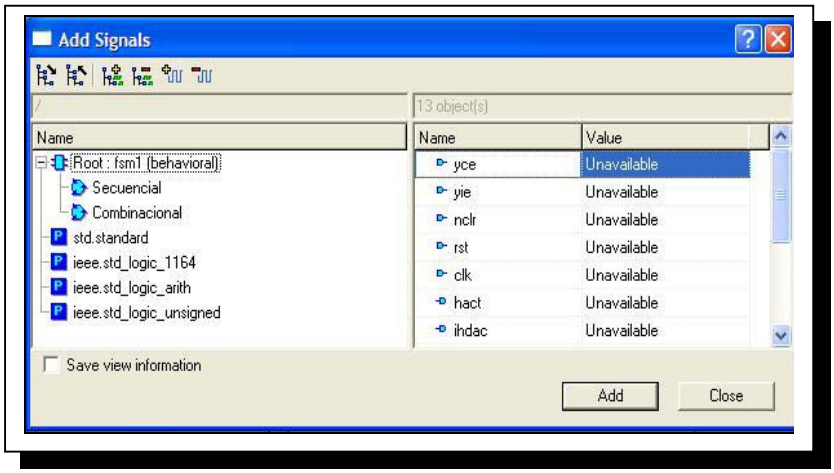


Fig. 4-44 Señales agregadas.

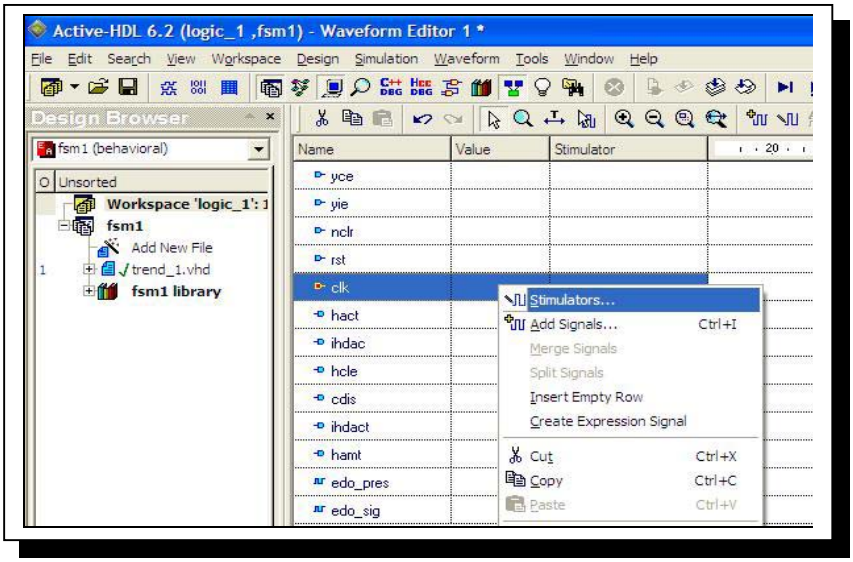


Fig. 4-45 Agregar estímulos.

Básicamente emplearemos dos tipos de estímulos para efectuar la simulación: Un reloj (la maquina que describimos es síncrona –un solo reloj-), en el podemos ajustar la frecuencia y el ciclo de trabajo, una vez ajustada la frecuencia damos clic en "*Apply*" y a continuación en "*Close*" (figura 4-46).

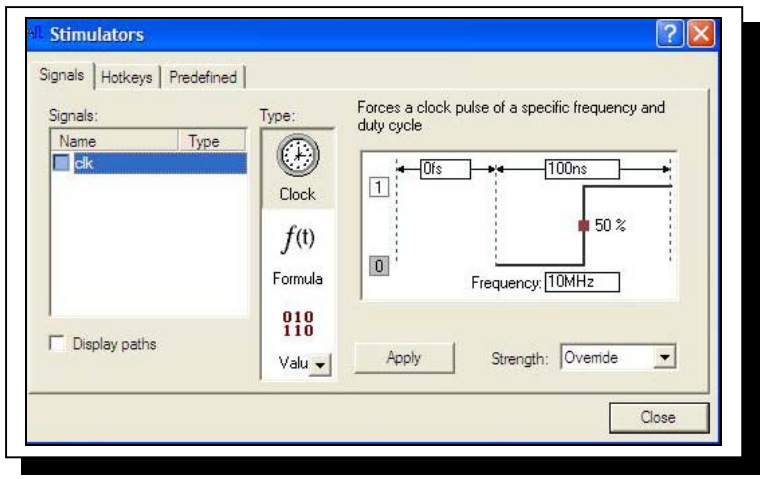


Fig. 4-46 Estimulo de reloj con frecuencia ajustable.

Y para el resto de las entradas usaremos un estimulo del tipo "*HotKey*", que no es otra cosa, sino que, a cada entrada se le asigna una letra del teclado con la que, una vez iniciada la simulación, es posible cambiar el estado entre 0 y 1 pulsando la tecla, lo cual resulta de particular utilidad (figura 4-47).

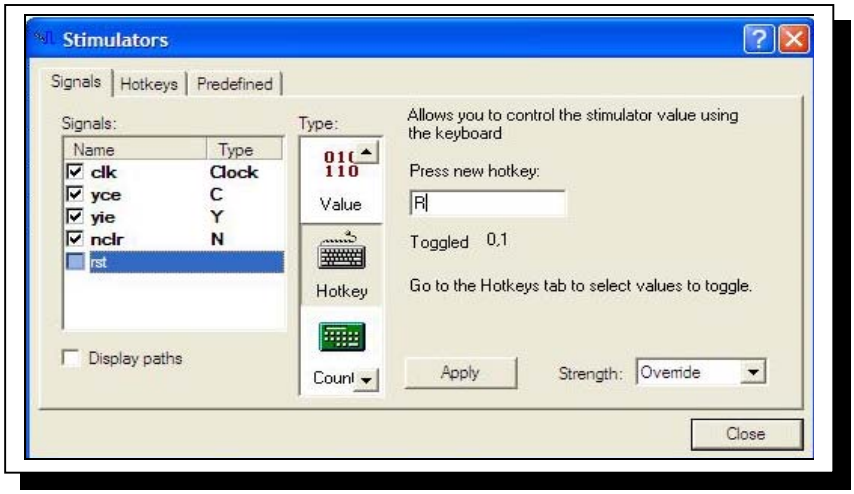


Fig. 4-47 Estimulo tipo "Hotkey"

4.27 Arrancar la simulación

Terminada la asignación de estímulos es posible arrancar la simulación de nuestro diseño. Se puede efectuar con cualquier de los dos iconos siguientes:

Simula hasta.. Con esta opción despliega un campo en el que se indica el periodo de tiempo de simulación a ejecutarse.

Simula por. Con esta otra opción se ejecutara la simulación durante un ciclo de reloj.

En el caso particular; resulta conveniente hacer el análisis en cada ciclo de reloj.

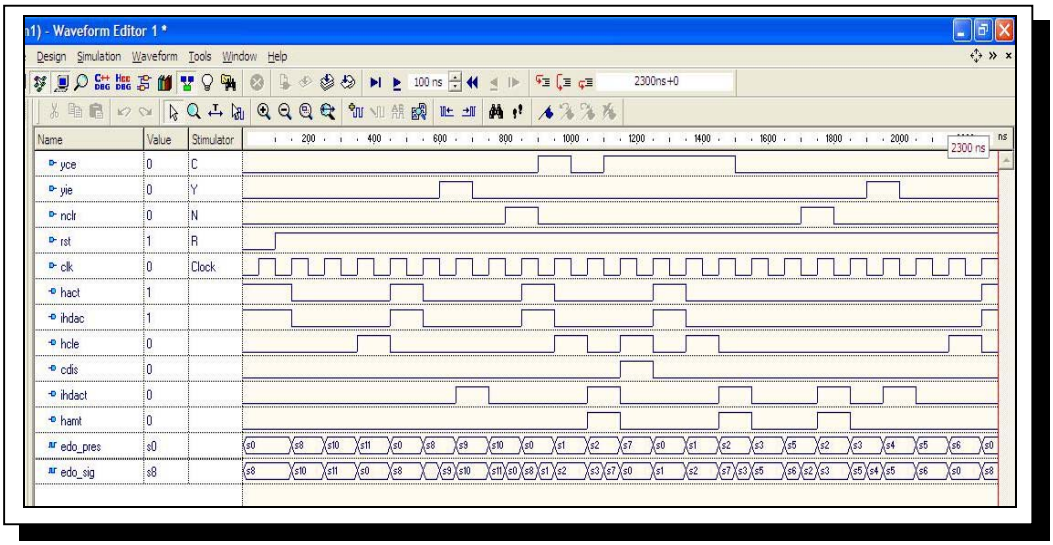
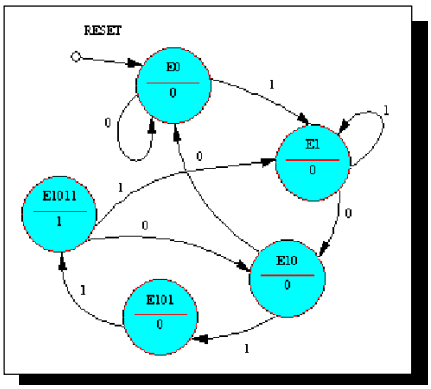


Fig. 4-48 Termino de simulación.

Finalmente, es posible obtener una grafica de tren de pulsos comportamental ("Behavioral") de nuestro diseño digital, para que una vez que la simulación muestre el comportamiento que deseamos de nuestro diseño podamos dar paso a la sintesis, implementacion y programacion en la tarjeta Saprtan 3E (figura 4-48).



5

Ejemplos

En esta seccion se han de desarrollar tres ejemplos de aplicación de sistemas digitales, todos ellos fueron tomados del temario de la clase de Diseño de Sitemas Digitales. Se hizo asi en virtud de los objetivos planteados en principio para el presente trabajo.

5.1 Planteamiento de un diagrama de flujo

El diagrama de flujo es una las técnicas de modelado mas utilizada en el diseño de los sistemas digitales dado qué:

- Es una técnica de análisis de sistemas.
- Representa gráficamente los procedimientos o etapas principales del proceso.
- Es un gráfico qué provee las etapas lógicas para la solución de un problema.
- Es una herramienta utilizada para mostrar una secuencia de acciones y decisiones de una forma lógica y sistemática.
- Utiliza símbolos qué representan máquinas, documentos o acciones a tomar durante el proceso o únicamente ciertas partes lógicas en el proceso.
- En los computadores, se puede utilizar para representar el flujo de información entre los distintos componentes del sistema de proceso de datos.
- El diagrama de flujo no informa de cómo se realizan ciertos procesos. Muestra quién (o qué equipo) hace qué y cuándo dentro de la operación de proceso de datos.

5.2 Niveles de los diagramas de flujo

Diagramas de flujo a nivel de concepto

- Utiliza sólo cajas rectangulares unidas por flechas o líneas de flujo.
- Sólo representa las principales etapas implicadas en la solución del problema.
- No se indican detalles ni puntos de decisión.

Diagramas de flujo a nivel de algoritmo

- Es más especifico.
- Muestra de manera más clara la lógica de la secuencia de acciones a realizar.
- Puede utilizar cualquiera de los símbolos qué se emplean para elaborar diagramas de flujo.
- Solo representa la lógica implicada en la resolución del problema.
- No describe etapas especificas o procesos qué debe de realizar un cierto µControlador.

Diagramas de flujo a nivel de instrucción

- Es utilizado para desarrollar programas de circuitos basados en µprocesadores.
- Proporciona información detallada del proceso a realizar.
- Se elaboran teniendo en cuenta la arquitectura interna del µprocesador y su juego de instrucciones.
- Suministra un juego detallado de procedimientos, a partir de los cuales se pueden codificar programas.

SÍMBOLO	NOMBRE	DESCRIPCIÓN
	Proceso	Utilizado para indicar una tarea función importante a realizar.
	Línea de flujo	Indica la dirección de la secuencia de acciones.
	Terminal	Utilizado para indicar el inicio fin de un proceso o secuencia de acciones
	Conector	Se emplea para indicar la continuación de la secuencia.
	Decisión	Este denota una bifurcación condicional en referencia a una relación de magnitudes; o bien a la existencia o no de una condición especifica.
	Subrutina	Corresponde a una misma secuencia de acciones qué se usa varias veces en el proceso.

Fig. 5-1 Simbolos basicos en la construccion de diagramas de flujo.

Los símbolos básicos para elaborar un diagrama de flujo se muestran en la figura 5-1.

5.3 La carta ASM

Este paso consiste en obtener una carta ASM (“Algorithmic State Machine”). En ella se agrega informacion añadiendo un tipo de elementos, los cuales se describen mas

adelante. Antes hay que mencionar las características principales de una carta ASM, las cuales son:

- Se asemeja a un diagrama de flujo.
- Es un diagrama especializado de estados finitos.
- Describe el comportamiento de una máquina digital secuencial.
- Puede estar formada por varias unidades de control y una ruta de datos.
- Se relaciona estrechamente con el diagrama de estados.
- Se elabora siguiendo un conjunto de reglas sencillas pero precisas.
- Contiene, en sus símbolos parecidos a cajas, una descripción condensada de las acciones y decisiones del sistema en función de las señales de control y de estado.

5.4 Elementos de la carta ASM

Cajas de estado o proceso

Es un rectángulo en el qué se especifica un conjunto de acciones, procesos o tareas qué se llevarán a cabo en un ciclo de reloj (figura 5-2). Representa un estado de control de la máquina digital.

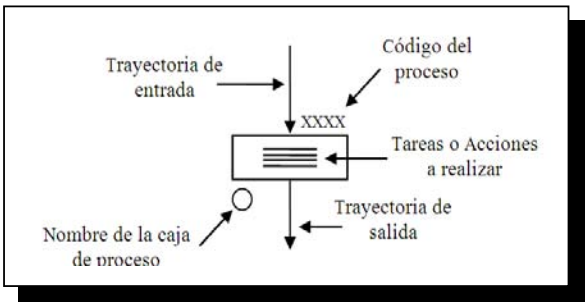


Fig. 5-2 Caja de Proceso

Caja de decisión

Es un rombo con una entrada y dos salidas. Contiene el nombre de una variable lógica qué sirve como condición qué se evaluará (figura 5-3). Cada ruta de salida de esta caja conduce a una caja de estado o a otra caja de decisión.

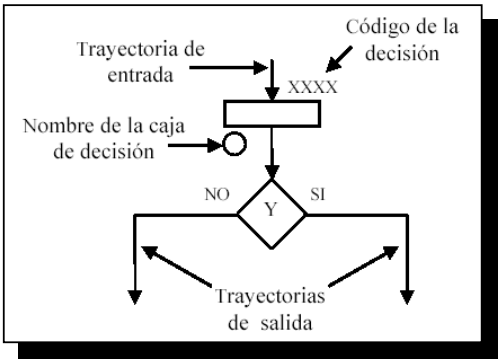


Fig. 5-3 Caja de decisión.

Caja de salida condicional

Es un rectángulo con lados redondeados. Su punto de entrada debe conectarse a una salida de una caja de decisión qué verifica alguna condición. La caja de salida condicional especifica las acciones qué se llevarán a cabo, únicamente si la condición se cumple en el ciclo de reloj actual, (figura 5-4).

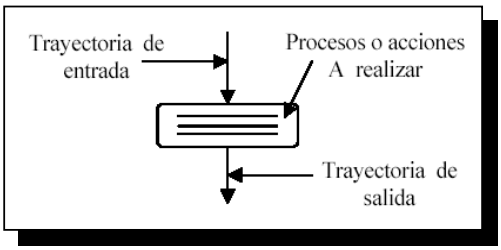


Fig. 5-4 Caja de salida condicional.

5.5 Diagrama de estados

A partir de la Carta ASM obtendremos el diagrama de estados correspondiente. Acerca de los diagramas de estado, se puede decir lo siguiente:

- Es una representación gráfica del comportamiento deseado de la secuencia de acciones y decisiones de un sistema digital.
- Muestra las posibles transiciones entre estados.

- Esta conformado por círculos u óvalos (nodos) qué representan todos los estados internos y flechas entre los nodos para representar transición entre estados.
- La información proporcionada corresponde a cada combinación de entrada primaria/estado presente; así como la correspondiente al estado siguiente/salida primaria.
- Proporciona la secuencia de entrada/salida del circuito en forma implícita y finita.

5.6 Elementos del diagrama de estados

Los elementos del diagrama de estados se muestran en la figura 5-5.

5.7 Diseño de una Máquina de Estados

VHDL permite realizar descripciones algorítmicas de alto nivel de máquinas de estados. De esta forma, el diseñador se evita tareas como generar la tabla de transición de estados o la obtención de las ecuaciones de excitación basadas en tipo de biestable.

SÍMBOLO	NOMBRE	DESCRIPCIÓN
	Circulo u ovalo	Se utiliza para representar a cada estado de la secuencia de acciones qué debe seguir un sistema digital.
	Línea de secuencia	Se utiliza para representar la o flecha transición entre estados.
	Salidas	Indica qué en el estado en qué están presentes existe una acción a realizar.

Fig. 5-5 Elementos del diagrama de estados.

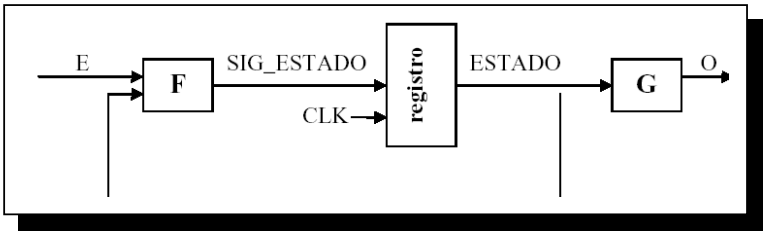


Fig. 5-6 Elementos del diagrama de estados.

Una Maquina de Estados Finita (FSM) se puede describir en VHDL de varias formas. En primer lugar en la sección de declaraciones de la arquitectura, se define un tipo numerado en el que se asignan identificadores a cada estado. Suele ser recomendable utilizar identificadores ilustrativos para los estados. La herramienta de síntesis será la encargada de codificar estos estados. Posteriormente, en el cuerpo de la arquitectura se define la función de transición de estados (F) y la función de salida (G) en uno o varios procesos, como se muestra en la figura 5-6.

Por lo tanto tenemos:

- Un proceso secuencial que modela los biestables de estado; Por lo tanto que actualiza el estado (ESTADO).

```
SINCRONO: process(clk,reset)
begin
if reset ='1' then
ESTADO<=S1;
elsif clk'event and clk='1' then
ESTADO<= SIG_ESTADO;
end if;
end process SINCRONO;
```

En este proceso intervienen las variables de la señal de reloj y de un *reset*. Si tiene lugar el evento *reset*, se ejecuta el estado inicial de la maquina de estados. Si tiene lugar el evento de reloj ("Flanco de subida") se actualiza el estado al próximo estado.

- Un proceso combinacional que modela las funciones F y G; por lo tanto deriva el siguiente estado (ESTADO_SIG) y actualiza las salidas (O).

```
CONTROL: process(ESTADO,E)
begin
case ESTADO is
when S1 =>
.
.
end case;
end process control;
end ARCH;
```

Para la implementación en VHDL. Primero definimos un tipo enumerado, formado por los nombres de los estados y se declaran dos señales de este tipo:

```
type ESTADOS is (S1, S2, S3, S4);
signal ESTADO, ESTADO_SIG: ESTADOS;
```

A continuación creamos un proceso combinacional en el que se determina el siguiente estado (ESTADO_SIG) y la salida S en función del estado actual .

```
when S1 =>
O <= '0';
if (E='0') then
SIG_ESTADO<=S2;
Else
SIG_ESTADO<=S1;
end if;
when S2 =>
```

5.8 Modelo de la maquina de estados.

El estilo de modelado que elegimos para efectuar la descripción de la maquina de estados del primer ejemplo es del tipo "Moore", por ser un tipo de maquina de estados finitos del tipo síncrono, significa que este tipo de maquinas depende del estado presente únicamente, estas maquinas contienen una única señal de reloj entre sus bloques. Este tipo de diseño es muy confiable, en la figura 5-7.

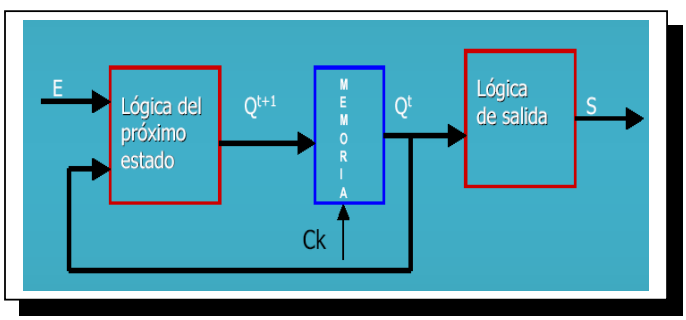


Fig. 5-7 Maquina de estados finitos tipo "Moore".

Una máquina de estados finitos tipo Moore es una máquina de estados que evoluciona síncronamente en función del estado y las entradas y cuyas salidas toman valores en función del estado en que se encuentra.

Si el estado de funcionamiento de un diseño, y por ende las salidas, están dadas por las entradas al circuito solamente el diseño puede resolverse mediante lógica combinacional. Un diseño en el que el estado de funcionamiento no está solo dado por las entradas sino también por la historia de funcionamiento anterior no puede resolverse mediante lógica combinacional. Se está presencia de un circuito secuencial que incluirá no solo lógica combinacional sino también una máquina de estados. Si la cantidad de estados es limitada a se la denomina máquina de estados finitas.

Siguiendo las recomendaciones para diseños sintetizables sobre FPGA's, se recomienda utilizar máquinas de estado sincrónicas, con un estado de reset asincrónico. En las máquinas de estado sincrónicas solo hay cambios de estado y sobre las salidas con un flanco de reloj. Si la velocidad del reloj es demasiado rápida y se desea hacer más lenta una máquina de estados, se puede utilizar una señal de habilitación de reloj en el proceso sincrónico.

5.9 Edición de la maquina de estados.

La edición la haremos en el editor de Active – Hdl, los pasos para efectuarlo están descritos claramente en el Capitulo 4 del presente trabajo.

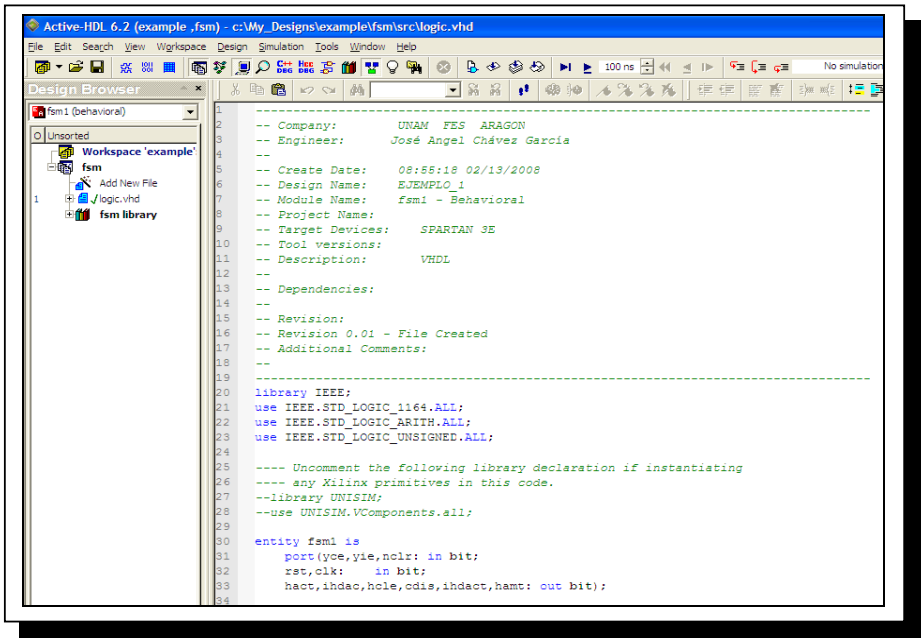


Fig. 5-8 Ventana de edición con Active - Hdl.

5.10 Simulación del sistema descrito.

Efectuamos los pasos para efectuar la simulación del sistema, el cual dará por resultado un "*trending*" como el que muestra la figura 5-9.

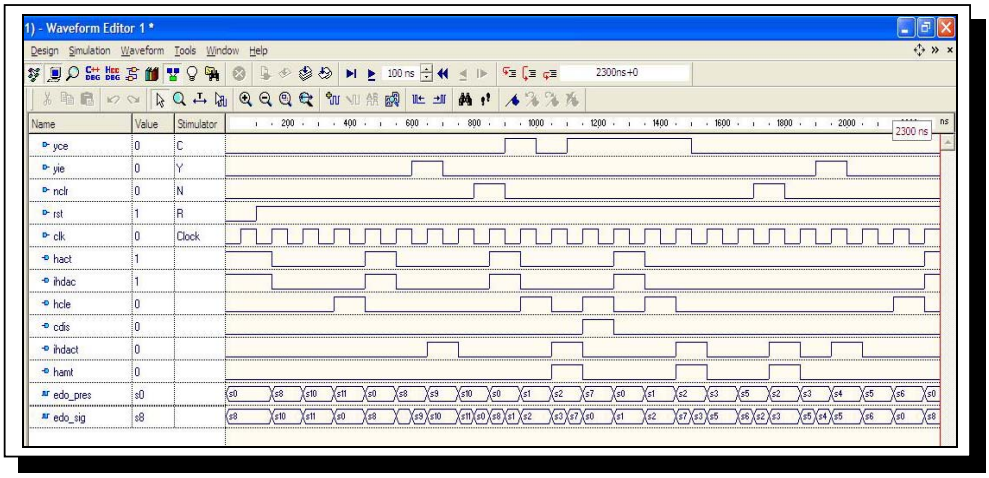


Fig. 5-9 Resultado de la simulación temporal.

Terminada la simulación. Corroboramos que el funcionamiento de la descripción que hicimos de la maquina de estados es la que deseamos.

5.11 Copiar el código fuente al entorno ISE Project Navigator.

Una vez que, la simulación ha arrojado un comportamiento adecuado a nuestro diseño, copiamos el código y lo trasladamos a un modulo HDL en el entorno del ISE Project Navigator (Referirse al Capitulo 4). Mostrándose una pantalla igual a la de la figura 4-10.

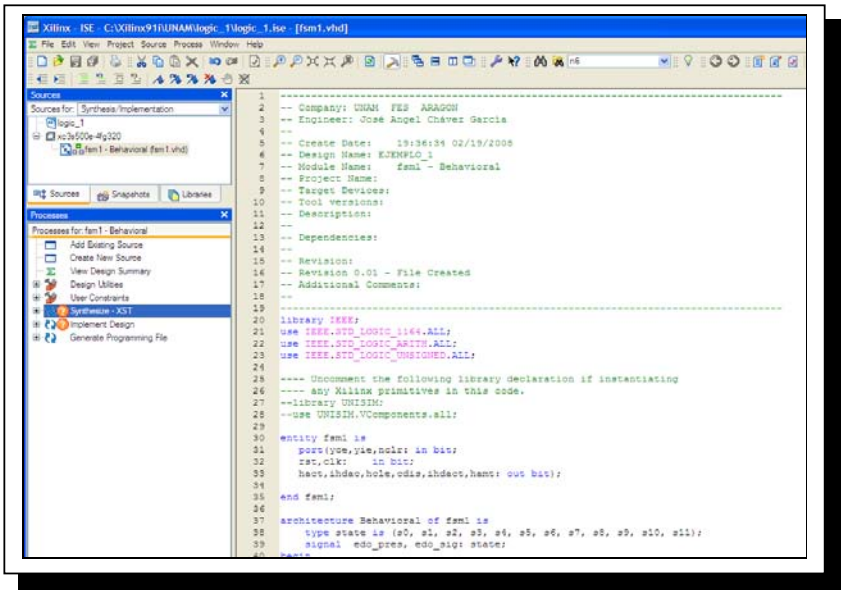


Fig. 5-10 Editor de ISE Project Navigator.

5.12 Síntesis, Implementación y programación.

Debemos entonces completar los pasos correspondientes al proceso de diseño que aparecen debidamente descritos en el Capitulo 4.

5.13 Comprobación física.

El ciclo de diseño lo completamos al estimular las entradas de la tarjeta Spartan y comprobar que se ejecutan adecuadamente las salidas, de acuerdo al diagrama de estados asociado.

5.14 Diagrama de flujo del ejemplo_1.

El primer diagrama de flujo describe un sistema digital de una maquina de las llamadas "tragamonedas". En la figura figura 5-11 se muestra este primer diagrama.

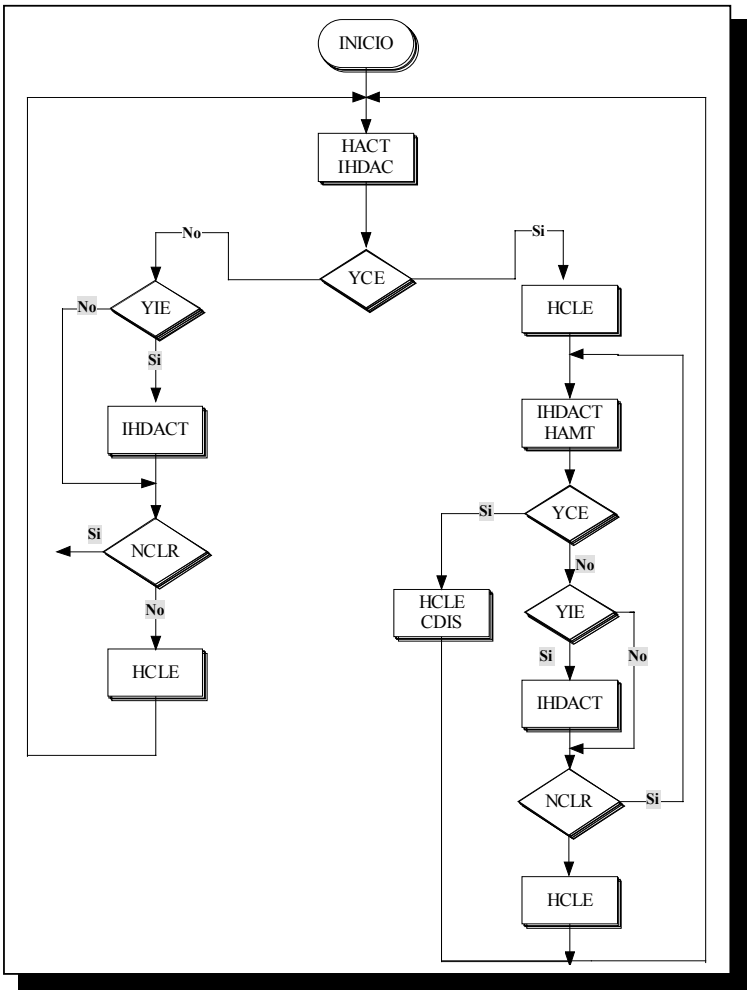


Fig. 5-11 Diagrama de flujo de la maquina tragamonedas.

5.15 Carta ASM del ejemplo_1.

Se redibuja el diagrama de estados y obtenemos la Carta ASM. Se En este ejemplo se insertan cuatro cajas de proceso en los estados (d), (f), (i) y (k) antes de las cajas de decision. Estas cajas no contienen tareas o acciones a realizar (figura 5-12).

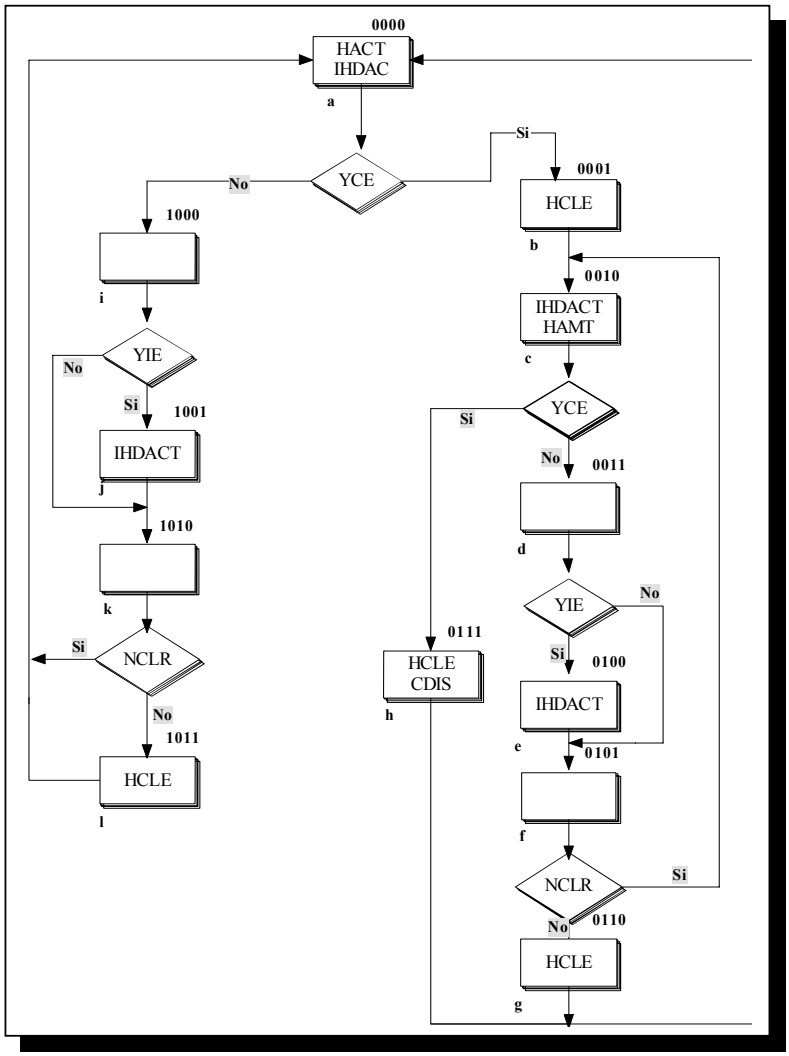


Fig. 5-12 Carta ASM del ejemplo_1.

5.16 Diagrama de estados del ejemplo_1.

La obtencion del diagrama de estados del primer ejemplo quedara como se muestra en la figura 5-13.

5.17 Descripción HDL del ejemplo_1.

Una vez que se ha obtenido el diagrama de estados; es posible describir en lenguaje HDL la maquina de estados, recordando que lo hacemos editando en el software Active – Hdl.

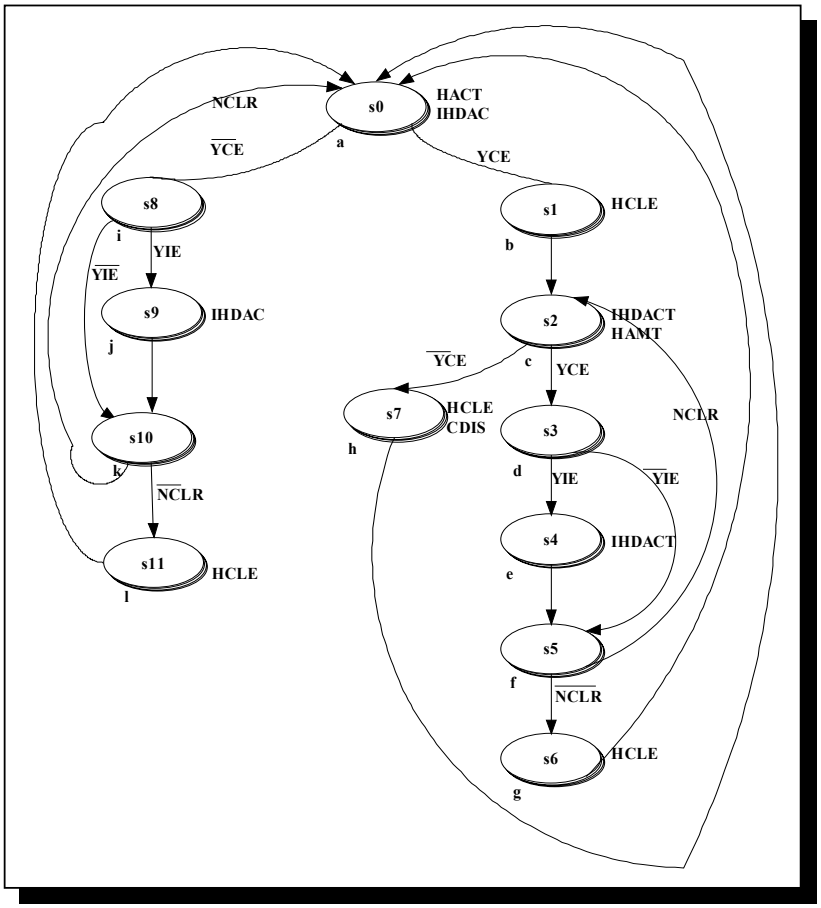


Fig. 5-13 Diagrama de estados del ejemplo_1.

En el apéndice A del presente trabajo; se anexa el código fuente completo de este primer ejemplo (Se hace la descripción en VHDL y en Verilog). En la figura 5-14, se muestra la pantalla de edición de Active.(Los pasos aparecen detallados en el Capitulo 4, sección 4.19 a 4.21)

5.18 Simulación del ejemplo_1.

Para cada ciclo de reloj y de acuerdo a los estímulos que tengan las entradas; verificamos que las salidas que deben activarse en cada estado y las señales de estado presente y estado siguiente sean las que corresponden. Se verifica en el primer ejemplo que tiene el comportamiento que deseamos (figura 5-15). Los pasos a desarrollar para la simulación están descritos en el Capitulo 4, sección 4.22 a 4.26.

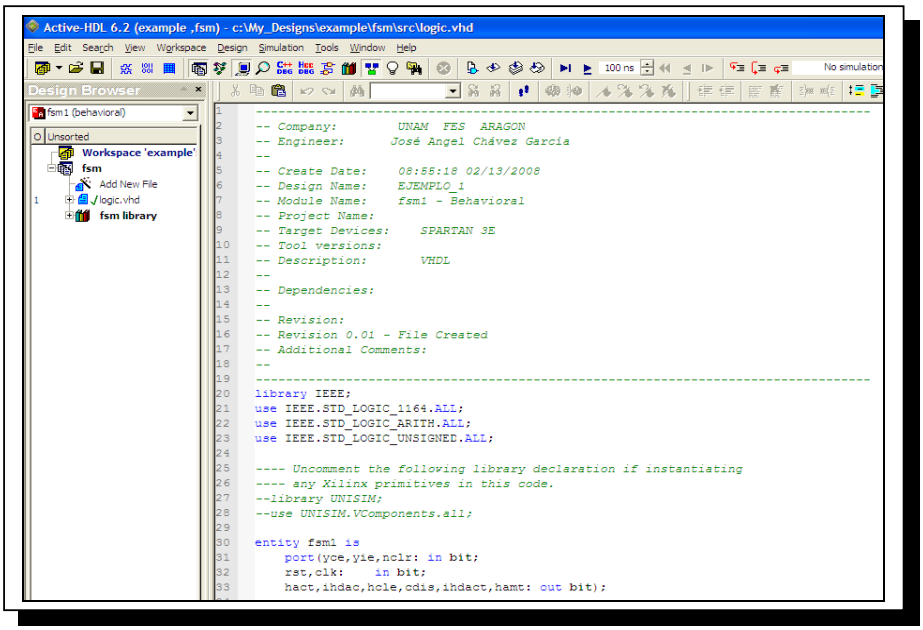


Fig. 5-14 Pantalla de edición de Active - Hdl

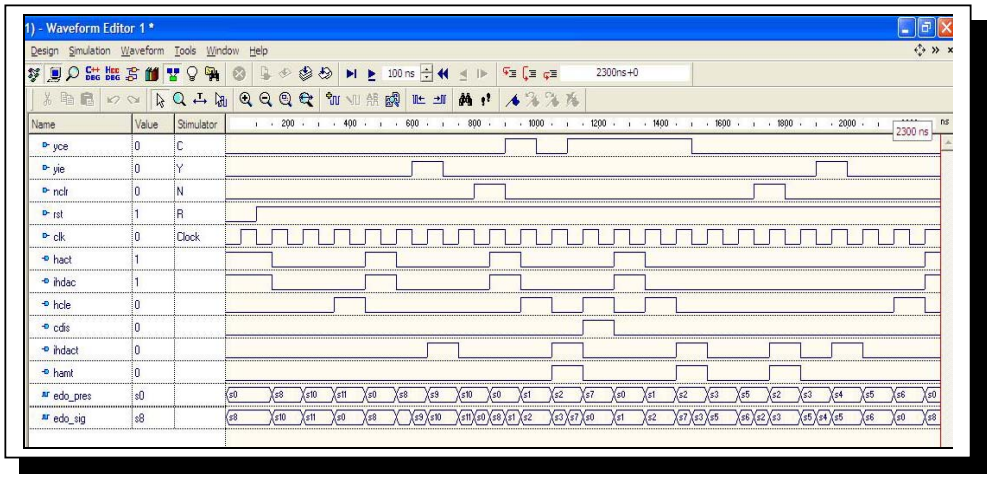


Fig. 5-15 Resultado de la simulación del ejemplo_1.

5.19 Síntesis, Implementación y programación del ejemplo_1

Efectuamos los pasos de síntesis, implementación y programación del ejemplo_1. Los pasos a desarrollar están descritos de forma clara en el Capitulo 4 (En la subseccion 4.5 a la 4.17). Una programación satisfactoria deberá mostrar la leyenda mostrada en la figura.

Fig. 5-16 Programacion satisafctoria.

5.20 Diagrama de flujo del ejemplo_2.

El diagrama de flujo de este ejemplo corresponde a una maquina grabadora de mensajes de un telefono.

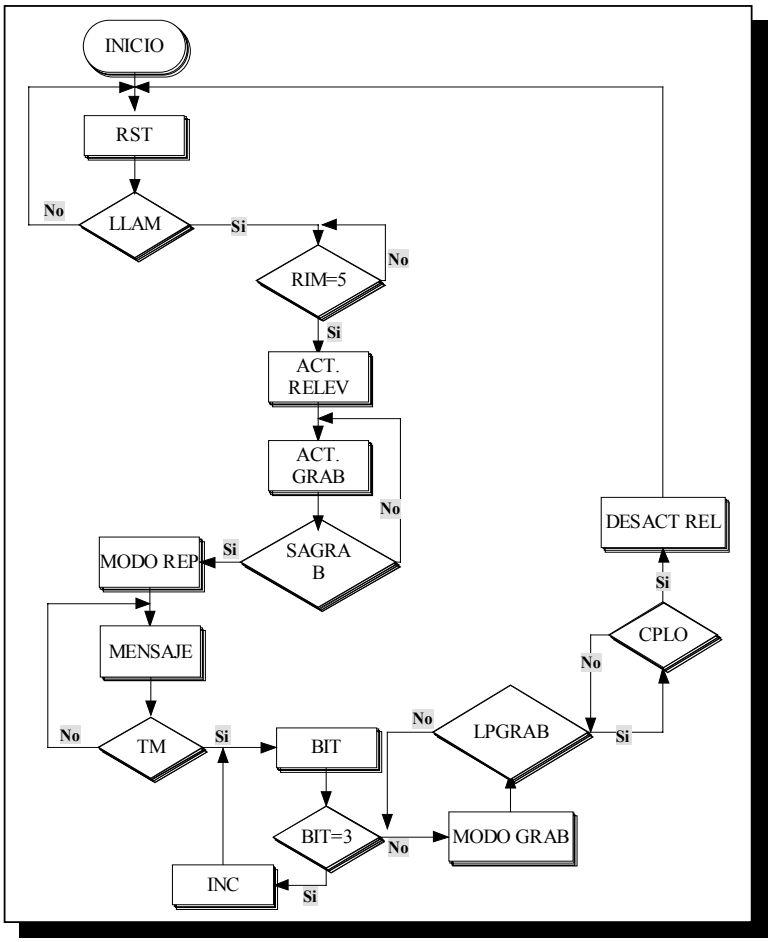


Fig. 5-17 Diagrama de flujo del ejemplo_2

5.21 Carta ASM del ejemplo_2.

Para este ejemplo, se agregan dos cajas de proceso en los estados (b) y (j). Efectuando la conversión completa, la carta será como se muestra en la figura 5-18.

5.22 Diagrama de estados del ejemplo_2.

El diagrama de estados obtenido para el ejemplo_2; será como el que se muestra a continuación en la figura 5-19.

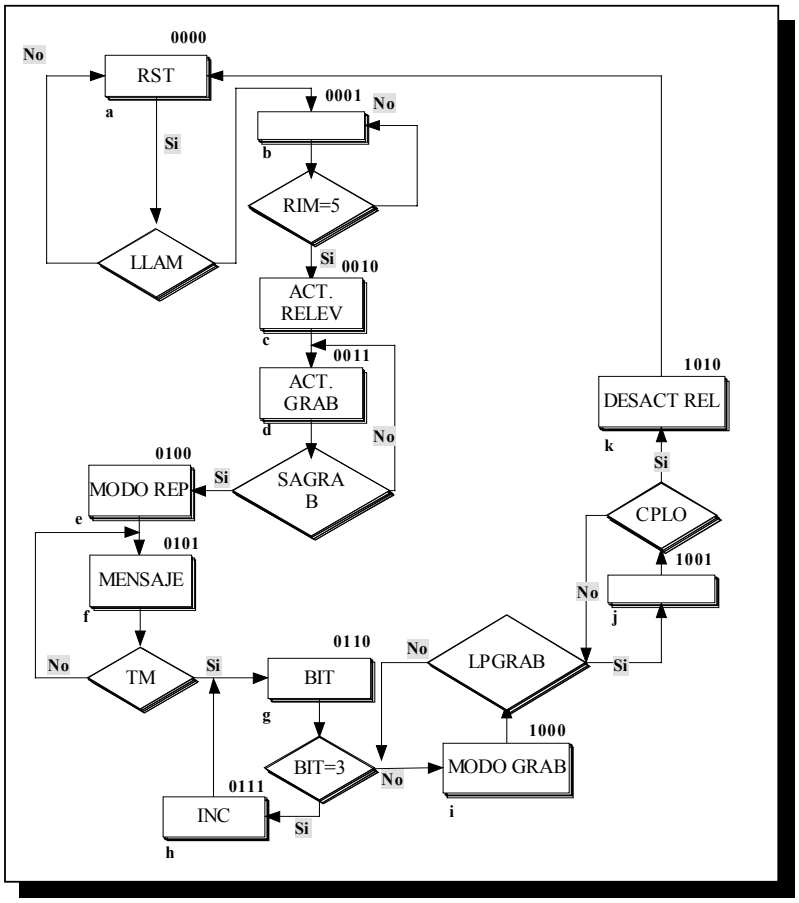


Fig. 5-18 Carta ASM del ejemplo_2.

5.23 Descripción HDL del ejemplo_2.

Una vez que se ha obtenido el diagrama de estados; es posible describir en lenguaje HDL la maquina de estados, editamos en el software Active – Hdl (figura 5-20). En el apéndice A del presente trabajo; se anexa el código fuente completo de este primer ejemplo (Se hace la descripción en VHDL). En la figura se muestra la pantalla de edición de Active. .(Los pasos aparecen detallados en el Capitulo 4, sección 4.19 a 4.21)

5.24 Simulación para el ejemplo_2.

Para cada ciclo de reloj y de acuerdo a los estímulos que tengan las entradas; verificamos que las salidas que deben activarse en cada estado y las señales de estado presente y estado siguiente sean las que corresponden (figura 5-21). En este segundo ejemplo comprobamos de igual forma que tiene el comportamiento que deseamos. Los pasos a desarrollar para la simulación están descritos en el Capitulo 4, sección 4.22 a 4.26.

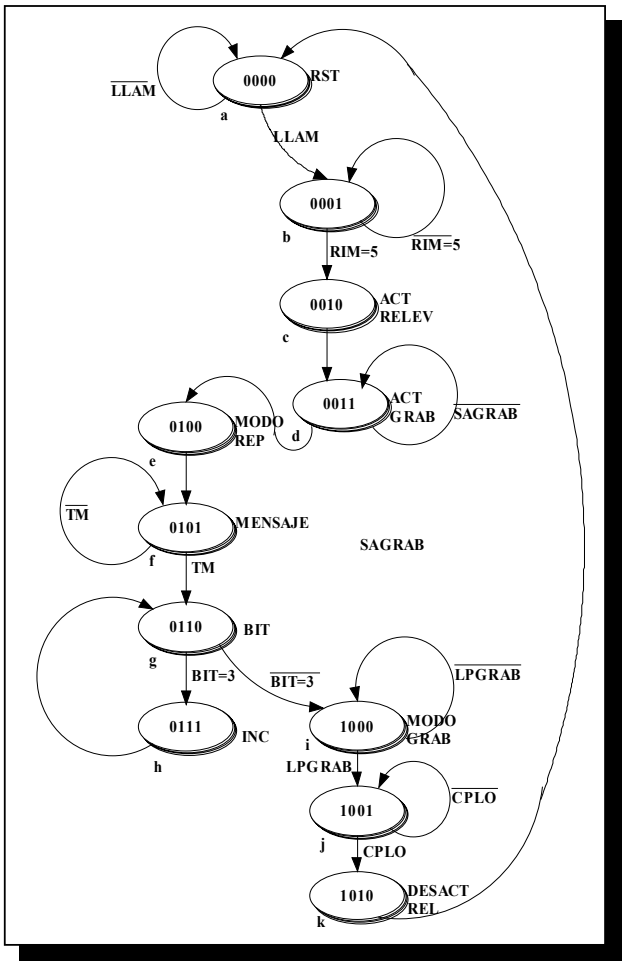


Fig. 5-19 Diagrama de estados del ejemplo_2.

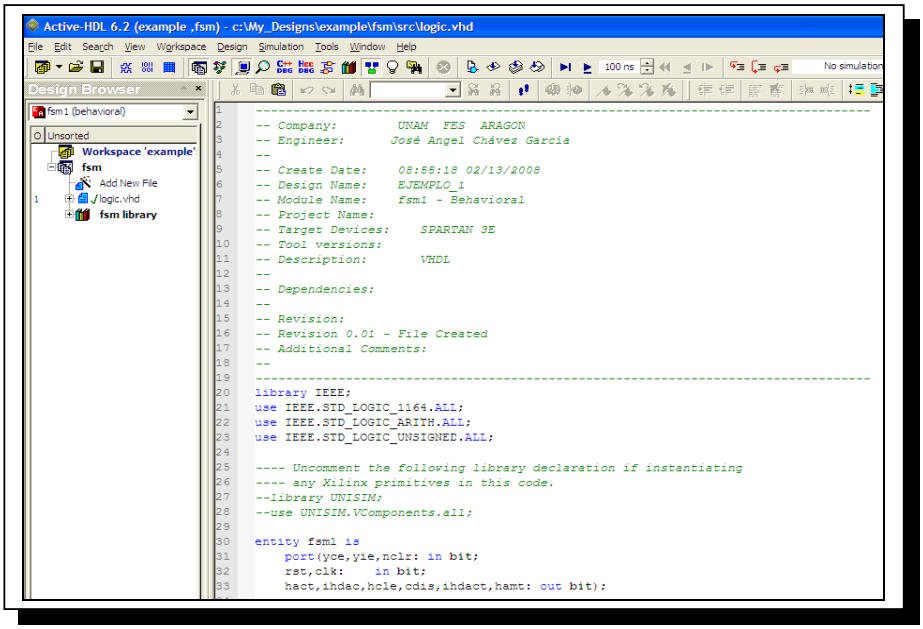


Fig. 5-20 Edición del ejemplo_2 en Active – Hdl.

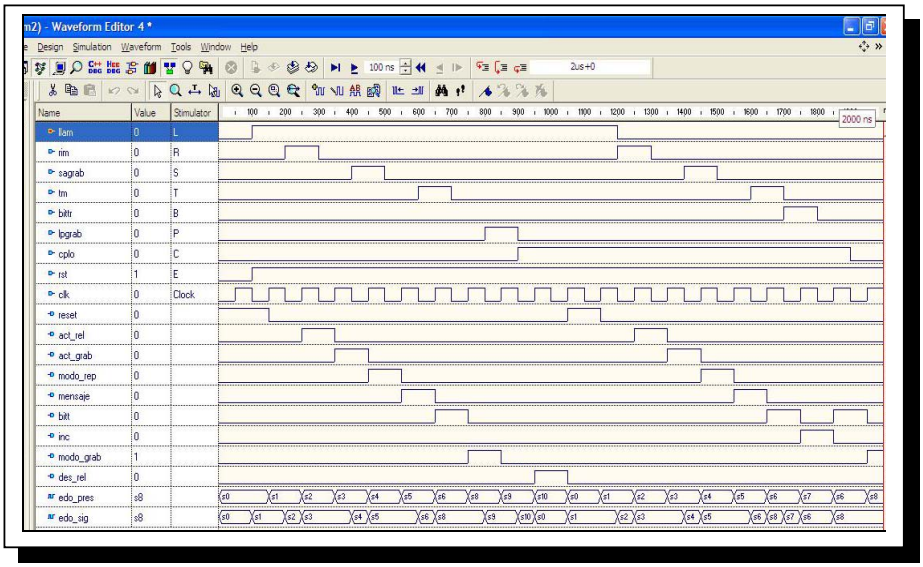


Fig. 5-21 Resultado de la simulación del ejemplo_2.

5.25 Síntesis, Implementación y programación del ejemplo_2

Efectuamos los pasos de síntesis, implementación y programación del ejemplo_1. Los pasos a desarrollar están descritos de forma clara en el Capitulo 4 sección 4.5 a 4.17. Una programación satisfactoria deberá mostrar la leyenda mostrada en la figura 5-22.

Fig. 5-22 Programación satisfactoria.

5.26 Diagrama de flujo del ejemplo_3.

El diagrama de flujo de este ejemplo corresponde a la logica de un cajero automatico.

5.27 Carta ASM del ejemplo_3.

En el caso de este ejemplo, se agregan cinco cajas de proceso en los estados (c), (d), (e), (h) y (j), sin tareas o instrucciones a realizar. Efectuando la conversión completa, la carta será como se muestra en la figura 5-24.

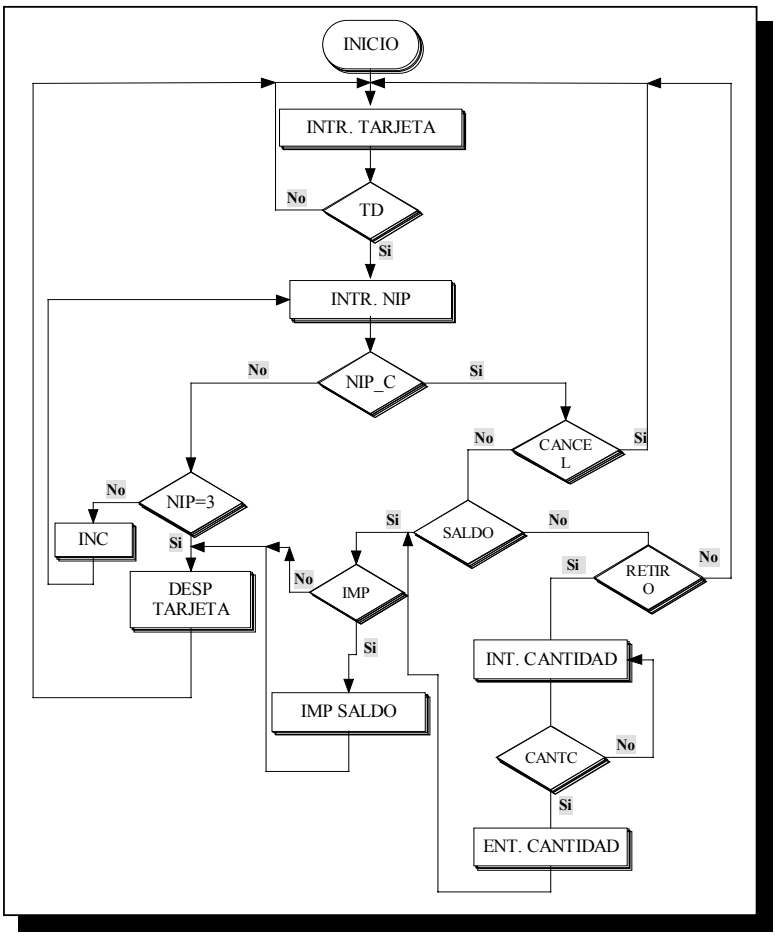


Fig. 5-23 Diagrama de flujo del ejemplo_3.

5.28 Diagrama de estados del ejemplo_3.

El diagrama de estados obtenido para el ejemplo_3 será como el que se muestra en la figura 5-25.

5.29 Descripción HDL del ejemplo_3.

Una vez que se ha obtenido el diagrama de estados; es posible describir en lenguaje HDL la maquina de estados, editamos en el software Active – Hdl (figura 5-26). En el apéndice A del presente trabajo; se anexa el código fuente completo de este tercer ejemplo (Se hace la descripción en VHDL). En la figura se muestra la pantalla de edición de Active. .(Los pasos aparecen detallados en el Capitulo 4, sección 4.19 a 4.21).

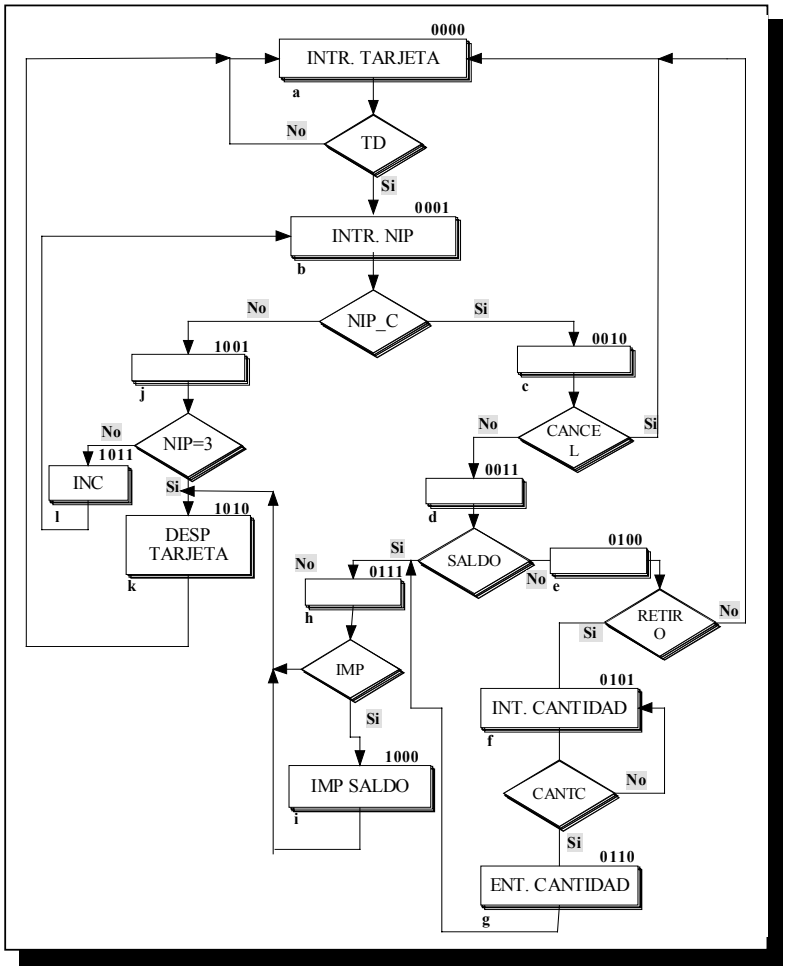


Fig. 5-24 Carta ASM correspondiente al ejemplo_3.

5.30 Simulación para el ejemplo_3.

Para cada ciclo de reloj y de acuerdo a los estímulos que tengan las entradas; verificamos que las salidas que deben activarse en cada estado y las señales de estado presente y estado siguiente sean las que corresponden. En este tercer y último ejemplo comprobamos de igual forma que tiene el comportamiento que deseamos (figura 5-27). Los pasos a desarrollar para la simulación están descritos en el Capitulo 4, sección 4.22 a 4.26.

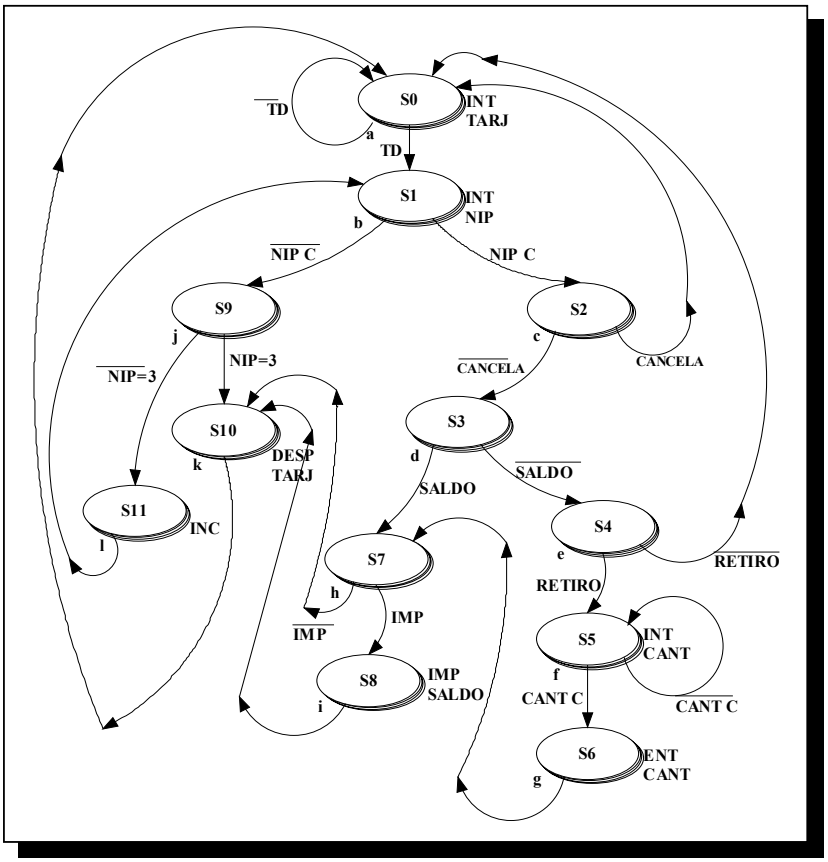


Fig. 5-25 Diagrama de estados del ejemplo_3.

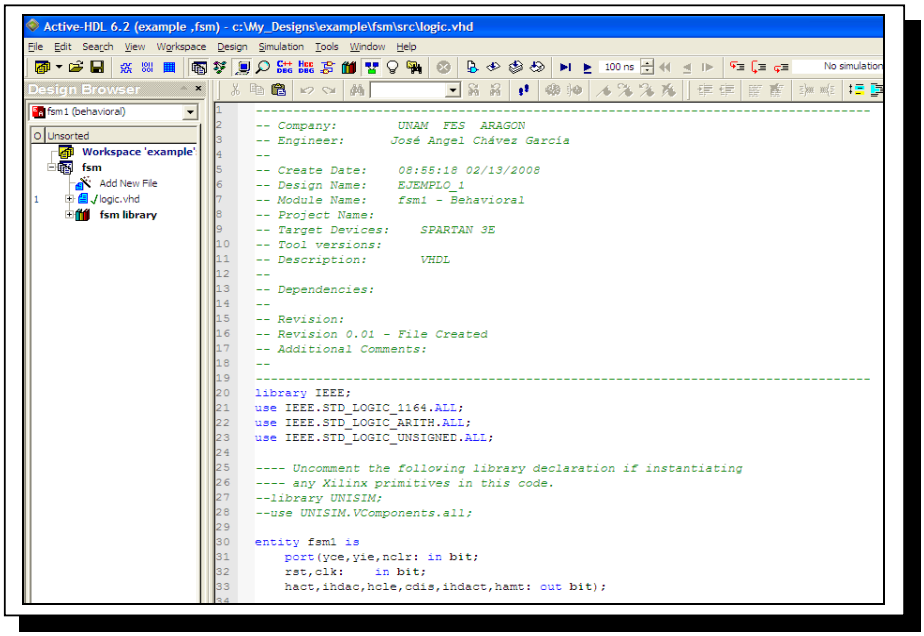


Fig. 5-26 Edición del ejemplo_3 en Active – Hdl.

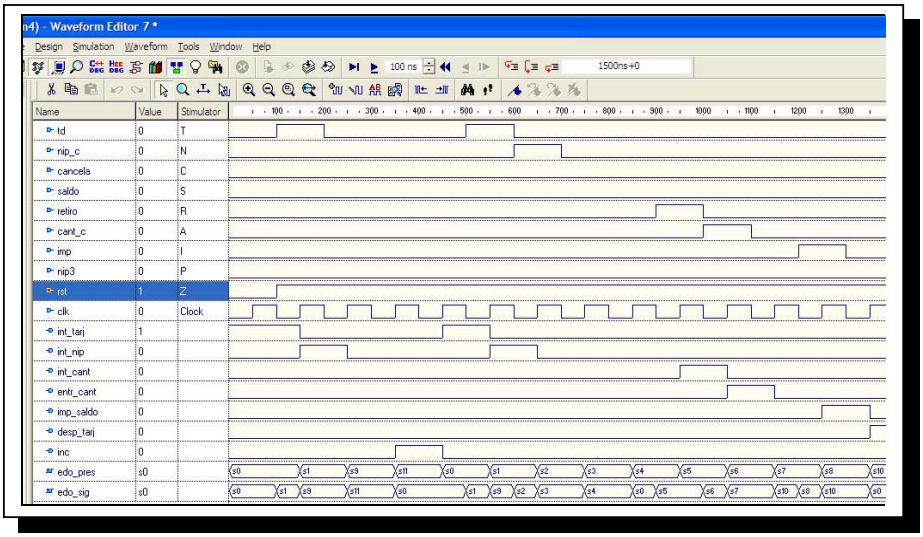


Fig. 5-27 Resultado de la simulación del ejemplo_3.

5.31 Síntesis, Implementación y programación del ejemplo_3

Efectuamos los pasos de síntesis, implementación y programación del ejemplo_3. Los pasos a desarrollar están descritos de forma clara en el Capitulo 4 sección 4.5 a 4.17. Una programación satisfactoria deberá mostrar la leyenda mostrada en la figura 5-28.

Fig. 5-28 Programación satisfactoria.

CONCLUSIONES

En el desarrollo del presente trabajo me pude introducir al manejo de las herramientas de diseño digital; observando y constatando que tienen posibilidades importantes de aplicación.

Lo anterior, en el caso particular de conseguir el cumplimiento de los objetivos planteados en un principio para la elaboración de mi trabajo de tesis.

Estos objetivos fueron conseguidos de manera satisfactoria; por medio del empleo de lenguajes de descripción de circuitos (HDL's); software de simulación y síntesis, así como aplicando las metodologías de diseño que se imparten en la Materia de Diseño de Sistemas Digitales de la FES Aragón.

El hecho de haber empleado un dispositivo de tecnología electrónica de última generación me permitió asimilar conceptos de utilidad para mi desarrollo profesional. Lo cual debe coadyuvar a mi formación integral como Ingeniero electrónico.

Debo resaltar que, el empleo y análisis de las herramientas de simulación utilizando el criterio adecuado nos lleva a ahorrar gran tiempo en el proceso de diseño de los sistemas digitales.

Puntualizar además cuan importante es el aprendizaje y conocimiento de las técnicas de descripción de sistemas digitales, caso particular de las maquinas de estados y los tipos que existen de estas ultimas.

Es un deseo personal que el presente pueda servir de referencia o de ayuda si llega a ser consultado; desde mi punto de vista pueden desprenderse trabajos de desarrollo que muestren, por ejemplo, la complejidad de sistemas digitales que puede llegar a implementarse en un dispositivo FPGA, por mencionar uno.

Considero, en términos generales que todo aquel profesionista debe estar en una búsqueda de adquisición de conocimiento de manera continua. Asimilando y empleando las nuevas técnicas; al igual que los dispositivos tecnológicos de ultima generación.

Finalmente debo anotar que, la elaboración de esta obra intelectual no estuvo excenta de dificultades y dudas; todas las cuales al haber superado me han generado un acervo que aquilato sobremanera; y que al concluir el presente me llena de satisfacción.

APENDICE A

Código Fuente

1. Código fuente en VHDL de la maquina de estados del primer ejemplo: FSM1.

```
----------------------------------------------------
-- Company: UNAM FES ARAGON
-- Engineer: José Angel Chávez García
--
-- Create Date: 19:36:34 02/19/2008
-- Design Name: EJEMPLO_1
-- Module Name: fsm1 - Behavioral

----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fsm1 is
        port(yce,yie,nclr: in bit;
        rst,clk:  in bit;
        hact,ihdac,hcle,cdis,ihdact,hamt: out bit);

end fsm1;

architecture Behavioral of fsm1 is
         type state is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11);
         signal  edo_pres, edo_sig: state;
begin
Secuencial: process (rst,clk)
begin
        if(rst='0') then
                edo_pres <= s0;

        elsif(clk'event and clk='1') then
                edo_pres <= edo_sig;
        end if;
        end process;

Combinacional: process(edo_pres, yce, yie, nclr)
begin
        hact <= '0';
        ihdac <= '0';
        hcle <= '0';
        cdis <= '0';
        ihdact <= '0';
        hamt <= '0';

case edo_pres is

        when s0 => hact <= '1';  ihdac <= '0';

        if(yce = '1') then
                edo_sig <= s1;
        else
                edo_sig <= s8;
        end if;

        when s1 => hcle <= '1';

        edo_sig <= s2;

        when s2 =>ihdact <= '1'; hamt <='0';

        if(yce = '0') then
                edo_sig <= s3;
        else
                edo_sig <= s7;
        end if;

        when s3 =>

        if(yie = '1') then
                edo_sig <= s4;
        else
                edo_sig <= s5;
        end if;

        when s4 => ihdact <= '1';

        edo_sig <= s2;

        when s5 =>

        if(nclr= '1') then
                edo_sig <= s2;
        else
                edo_sig <= s6;
        end if;
```

```
        when s6 => hcle <='1';

                edo_sig <= s0;

        when s7 =>hcle <='1'; cdis <= '1';

                edo_sig <= s0;

        when s8 =>

        if(yie= '1') then
                edo_sig <= s9;
        else
                edo_sig <= s10;
        end if;

        when s9 => ihdact <= '1';
                edo_sig <= s10;

        when s10 =>

        if(nclr= '1') then
                edo_sig <= s0;
        else
                edo_sig <= s11;
        end if;

        when s11 =>hcle <= '1';

        edo_sig <= s0;

        end case;
        end process;
end Behavioral;
```

2. Código fuente en VHDL de la maquina de estados del segundo ejemplo: FSM2.

```
----------------------------------------------------------
-----------------------
-- Company:  UNAM  FES  ARAGON
-- Engineer:  José Angel Chávez García
--
-- Create Date:    18:11:46 02/09/2008
-- Design Name: EJEMPLO_2
-- Module Name:    fsm2 - Behavioral

----------------------------------------------------------
-----------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
---- Uncomment the following library
declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fsm2 is

port( llam, rim, sagrab, tm, bittr, lpgrab, cplo: in
bit;
                rst, clk:               in bit;
                reset,      act_rel,      act_grab,
modo_rep: out bit;
                mensaje, bitt, inc, modo_grab,
des_rel:  out bit);

end fsm2;

architecture Behavioral of fsm2 is
                type STATE is (s0, s1, s2, s3,
s4, s5, s6, s7, s8, s9, s10);
                signal    edo_pres,    edo_sig:
STATE;
begin

SEQ: process(rst,clk)
begin

                if(rst = '0') then
                edo_pres <= s0;
                elsif(clk'event and clk = '1')
then
                edo_pres <= edo_sig;
                end if;

end process;


COMB: process(edo_pres, llam, rim, sagrab,
tm, bittr, lpgrab, cplo)
begin

        reset <= '0';
        act_rel <= '0';
        act_grab <= '0';
        modo_rep <= '0';
        mensaje <= '0';
        bitt <= '0';
        inc <= '0';
        modo_grab <= '0';
        des_rel <= '0';


case edo_pres is


        when s0 => reset <= '1';

                if(llam = '1') then
```

```
edo_sig <= s1;

                else
                edo_sig <= s0;

                end if;

        when s1 =>

                if(rim = '1') then
                edo_sig <= s2;

                else
                edo_sig <= s1;

                end if;

        when s2 => act_rel <= '1';
          edo_sig <= s3;

        when s3 => act_grab <= '1';

                if(sagrab = '1') then
                edo_sig <= s4;

                else
                edo_sig <= s3;

                end if;

        when s4 => modo_rep <= '1';

                        edo_sig <= s5;

        when s5 => mensaje <= '1';

                if(tm = '1') then
                edo_sig <= s6;

                else
                edo_sig <= s5;

                end if;
```

```
        when s6 => bitt <= '1';

                if(bittr = '1') then
                edo_sig <= s7;

                else
                edo_sig <= s8;

                end if;

        when s7 => inc <= '1';

                edo_sig <= s6;

        when s8 => modo_grab <= '1';

                if(lpgrab = '1') then
                edo_sig <= s9;

                else
                edo_sig <= s8;

                end if;

        when s9 =>

                if(cplo = '1') then
                edo_sig <= s10;

                else
                edo_sig <= s9;

                end if;

        when s10 => des_rel <= '1';

                edo_sig <= s0;


end case;
end process;
end Behavioral;
```

4. Código fuente en VHDL de la maquina de estados del tercer ejemplo: FSM3.

```
        ----------------------------------------------
        ----------------------------------
        -- Company:  UNAM  FES  ARAGON
        -- Engineer:  José Angel Chávez García
        --
        -- Create Date:    13:19:11 02/11/2008

        -- Design Name: EJEMPLO_3
        -- Module Name:    fsm3 - Behavioral
        ----------------------------------------------
        ----------------------------------

        library IEEE;
```

```
        use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fsm4 is
port( td,nip_c,cancela,saldo,retiro: in bit;
```

```
                cant_c,imp,nip3,rst,clk:
        in bit;

        int_tarj,int_nip,int_cant,entr_cant:    out
bit;
                imp_saldo,desp_tarj,inc:
        out bit);

end fsm4;

architecture Behavioral of fsm4 is
                type            state           is
(s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11);
                signal edo_pres,edo_sig: state;
begin

Secuencial: process(rst,clk)
begin
        if(rst='0') then
        edo_pres<=s0;
        elsif(clk'event and clk='1') then
        edo_pres<=edo_sig;
        end if;

        end process;

Combinacional:
process(edo_pres,td,nip_c,cancela,saldo,retiro,c
ant_c,imp,nip3)
begin

        int_tarj<='0';
        int_nip<='0';
        int_cant<='0';
        entr_cant<='0';
        imp_saldo<='0';
        desp_tarj<='0';
        inc<='0';


        case edo_pres is


when s0=> int_tarj<='1';

        if(td='1') then
        edo_sig<=s1;
        else
        edo_sig<=s0;
        end if;

when s1=> int_nip<='1';

        if(nip_c='1') then
        edo_sig<=s2;
        else
        edo_sig<=s9;
        end if;

when s2=>

        if(cancela='1') then
        edo_sig<=s0;
        else
        edo_sig<=s3;
        end if;

when s3=>

        if(saldo='1') then
        edo_sig<=s7;
        else
        edo_sig<=s4;
        end if;

when s4=>

        if(retiro='1') then
        edo_sig<=s5;
        else
        edo_sig<=s0;
        end if;


when s5=> int_cant<='1';

        if(cant_c='1') then
        edo_sig<=s6;
        else
        edo_sig<=s5;
        end if;

when s6=> entr_cant<='1';

        edo_sig<=s7;

when s7=>

        if(imp='1') then
        edo_sig<=s8;
        else
        edo_sig<=s10;
        end if;

when s8=> imp_saldo<='1';

        edo_sig<=s10;

when s9=>

        if(nip3='1') then
        edo_sig<=s10;
        else
        edo_sig<=s11;
        end if;

when s10=> desp_tarj<='1';

        edo_sig<=s0;
```

```
when s11=> inc<='1';

          edo_sig<=s0;
```

```
end case;
end process;
end Behavioral;
```

5. Código fuente en Verilog de la maquina de estados del primer ejemplo: FSM1.

```
// UNAM  FES  ARAGON
// José Angel Chávez G.
// FSM1   Verilog
module binary (clk, rst, yce, yie, nclr, hact, hcle,
ihdac, ihdact, cdis, hamt);


input clk, rst;

input yce, yie, nclr;

output hact, hcle, ihdac, ihdact, cdis, hamt;


reg hact, hcle, ihdac, ihdact, cdis, hamt;

// Declare the symbolic names for states

parameter [3:0]

   S1 = 4'b0001,
   S2 = 4'b0010,
   S3 = 4'b0011,
   S4 = 4'b0100,
   S5 = 4'b0101,
   S6 = 4'b0110,
   S7 = 4'b0111,
   S8 = 4'b1000,
   S9 = 4'b1001,
   S10 = 4'b1010,
   S11 = 4'b1011,
   S12 = 4'b1100;

// Declare current state and next state variables

reg [3:0] EP;
reg [3:0] ES;

// state_vector CS
   always @ (posedge clk or posedge rst)
   begin
      if (rst == 1'b1)
         EP = S1;
      else
         EP = ES;
   end
   always @(EP or yce or yie or nclr)
   begin
   case (EP)

                  S1 :
                  begin
         hact = 1'b1;
```

```
                  hcle = 1'b0;
                  ihdac = 1'b1;
                  ihdact = 1'b0;
                  cdis = 1'b0;
                  hamt = 1'b0;
      if (yce)
        ES = S2;
      else
        ES = S9;
      end

                  S2 :
                  begin

                  hact = 1'b0;
                    hcle = 1'b1;
                    ihdac = 1'b0;
                    ihdact = 1'b0;
                    cdis = 1'b0;
                    hamt = 1'b0;

        ES = S3;
      end
            S3 :
                  begin

                   hact = 1'b0;
                    hcle = 1'b0;
                    ihdac = 1'b0;
                    ihdact = 1'b1;
                    cdis = 1'b0;
                    hamt = 1'b1;

      if (yce)
        ES = S8;
      else
        ES = S4;
      end
                  S4 :
                  begin

                  hact = 1'b0;
                    hcle = 1'b0;
                    ihdac = 1'b0;
                    ihdact = 1'b0;
                    cdis = 1'b0;
                    hamt = 1'b0;
      if (yie)
        ES = S5;
      else
        ES = S6;
      end

                  S5 :
```

```
                    begin
                     hact = 1'b0;
                     hcle = 1'b0;
                     ihdac = 1'b0;
                     ihdact = 1'b1;
                     cdis = 1'b0;
                     hamt = 1'b0;

      ES = S6;
                     end
       S6 :
                      begin
                    hact = 1'b0;
                     hcle = 1'b0;
                     ihdac = 1'b0;
                     ihdact = 1'b0;
                     cdis = 1'b0;
                     hamt = 1'b0;

    if (nclr)
      ES = S3;
    else
      ES = S7;
    end
                    S7 :
                    begin
      hact = 1'b0;
                     hcle = 1'b1;
                     ihdac = 1'b0;
                     ihdact = 1'b0;
                     cdis = 1'b0;
                     hamt = 1'b0;

      ES = S1;
                     end

                     S8 :
                     begin
                      hact = 1'b0;
                     hcle = 1'b1;
                     ihdac = 1'b0;
                     ihdact = 1'b0;
                     cdis = 1'b1;
                     hamt = 1'b0;

      ES = S1;
                     end

                     S9 :
                     begin

      hact = 1'b0;
                     hcle = 1'b0;
                     ihdac = 1'b0;
                     ihdact = 1'b0;
                     cdis = 1'b0;
                     hamt = 1'b0;

      if (yie)
        ES = S10;
      else
        ES = S11;
      end

                    S10 :
                    begin
                    hact = 1'b0;
                     hcle = 1'b0;
                     ihdac = 1'b0;
                     ihdact = 1'b1;
                     cdis = 1'b0;
                     hamt = 1'b0;

       ES = S11;
       end

                     S11 :
                     begin
                     hact = 1'b0;
                     hcle = 1'b0;
                     ihdac = 1'b0;
                     ihdact = 1'b0;
                     cdis = 1'b0;
                     hamt = 1'b0;

      if (nclr)
        ES = S1;
      else
        ES = S12;
      end

                    S12 :
                    begin

       hact = 1'b0;
                     hcle = 1'b1;
                     ihdac = 1'b0;
                     ihdact = 1'b0;
                     cdis = 1'b0;
                     hamt = 1'b0;

       ES = S1;
                     end
              endcase

  end

endmodule
```

APENDICE B

Restricciones (“*Constraints*”)

Los siguientes, son ejemplos de restricciones comúnmente utilizadas en la tarjeta Spartan 3E Starter Kit.

```
##############################################################
### SPARTAN-3E STARTER KIT BOARD CONSTRAINTS FILE
##############################################################
# ==== Analog-to-Digital Converter (ADC) ====
# some connections shared with SPI Flash, DAC, ADC, and AMP
NET "AD_CONV" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
# ==== Programmable Gain Amplifier (AMP) ====
# some connections shared with SPI Flash, DAC, ADC, and AMP
NET "AMP_CS" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
# ==== Pushbuttons (BTN) ====
NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
# ==== Clock inputs (CLK) ====
NET "CLK_50MHZ" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
# Define clock period for 50 MHz oscillator (40%/60% duty-cycle)
NET "CLK_50MHZ" PERIOD = 20.0ns HIGH 40%;
# ==== Digital-to-Analog Converter (DAC) ====
# some connections shared with SPI Flash, DAC, ADC, and AMP
NET "DAC_CLR" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
# ==== 1-Wire Secure EEPROM (DS)
NET "DS_WIRE" LOC = "U4" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
# ==== Ethernet PHY (E) ====
NET "E_COL" LOC = "U6" | IOSTANDARD = LVCMOS33 ;
NET "E_CRS" LOC = "U13" | IOSTANDARD = LVCMOS33 ;
# ==== FX2 Connector (FX2) ====
NET "FX2_CLKIN" LOC = "E10" | IOSTANDARD = LVCMOS33 ;
# These four connections are shared with the J1 6-pin accessory header
NET "FX2_IO<1>" LOC = "B4" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
# These four connections are shared with the J2 6-pin accessory header
NET "FX2_IO<5>" LOC = "A6" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
# These four connections are shared with the J4 6-pin accessory header
NET "FX2_IO<9>" LOC = "D7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
# ==== PS/2 Mouse/Keyboard Port (PS2) ====
NET "PS2_CLK" LOC = "G14" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "PS2_DATA" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
# ==== Rotary Pushbutton Switch (ROT) ====
NET "ROT_A" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
NET "ROT_B" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP ;
NET "ROT_CENTER" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;
# ==== RS-232 Serial Ports (RS232) ====
NET "RS232_DCE_RXD" LOC = "R7" | IOSTANDARD = LVTTL ;
# Path to allow connection to top DCM connection
NET "SD_CK_FB" LOC = "B9" | IOSTANDARD = LVCMOS33 ;
# Prohibit VREF pins
CONFIG PROHIBIT = D2;
# ==== STMicro SPI serial Flash (SPI) ====
# some connections shared with SPI Flash, DAC, ADC, and AMP
NET "SPI_MISO" LOC = "N10" | IOSTANDARD = LVCMOS33 ;
NET "SPI_MOSI" LOC = "T4" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
# ==== Slide Switches (SW) ====
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
# ==== VGA Port (VGA) ====
NET "VGA_BLUE" LOC = "G15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST
;
NET "VGA_GREEN" LOC = "H15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
```

APENDICE C

GLOSARIO

ACE Agencia de certificación electrónica

ADC (*Analog to Digital Converter)* Acrónimo que significa Convertidor Analógico digital.
ASIC Circuito integrado de propósito especifico.
ASM. Maquina de estados algorítmica.
BIESTABLE Ver flip – flop
BYPASS Se evita ejecutar un paso a nivel software por considerar innecesario el mismo.
CAE Ingeniería asistida por computadora o por ordenador (*CAE*, del inglés *Computer Aided Engineering*) es el conjunto de programas informáticos que permiten analizar y simular los diseños de ingeniería realizados con el ordenador.
CPLD Es el acronimo de *Complex Proggramable logic device* Es la versión de alta densidad de integración y representa la evolucion de un PLD.
DAC *Digital to Analog Converter* Convertidor Digital Analógico.
DCE Conector Hembra para comunicación RS-232.
DTE Conector Macho para comunicación RS-232.
EMBEBIDO Se dice de un sistema en un solo chip; esto es, que esta "incluido".
FSM. Maquina de estados finitos.
ENCODER El encoder es un dispositivo electromecánico, que convierte la posición angular de su eje en una señal digital eléctrica.
FINITA Que es de una cantidad determinada o límitada.
FIRMWARE Software Controlador de un dispositivo electrónico.
FLIP FLOP Un biestable, también llamado báscula (*flip-flop* en inglés), es un multivibrador capaz de permanecer en un estado determinado o en el contrario durante un tiempo indefinido. Esta característica es ampliamente utilizada en electrónica digital para memorizar información.
FPGA (*field proggramable gate`s array)* Arreglo de compuertas de campo programables.
FULL DUPLEX Es una técnica de transmisión de datos capaz de, simultáneamente, transmitir y recibir datos.
HARDWARE Componentes físicos que integran un sistema que requiere además de programación.
HDL Hardware description languaje Lenguaje de descripción de hardware
HOTKEY Llamado a una rutina de menu de software por medio de una sola tecla.
IEEE Corresponde a las siglas de *The Institute of Electrical and Electronics Engineers*, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.
INTEL Es una empresa multinacional que fabrica microprocesadores, circuitos integrados especializados tales como circuitos integrados auxiliares para placas base de computadora y otros dispositivos electrónicos.
I/O Entradas/ Salidas.
ISE Integrated software environment Software de entorno integrado.
JTAG Acrónimo para *Joint Test Action Group*, es el nombre común utilizado para la norma IEEE 1149.1 utilizada para probar PCBs utilizando escaneo de límites.
JUMPER Elemento para interconectar dos terminales de manera temporal sin tener que efectuar una operación que requiera herramienta adicional, dicha unión de terminales cierran el circuito eléctrico del que forma parte.
LAYOUT En términos de programación, es cuando se modifica una pantalla, o sea el editor gráfico de algunos determinados lenguajes de programación.
LED Diodo emisor de luz.
LCD *Liquid cristal display* Pantalla de cristal liquido
MODEM Modulador / Demodulador, dispositivo que permite la transferencia de información de documentos en forma digital.

MULTIBOOT Es una especificación que define un protocolo entre los gestor de arranque y los kernels de los sistemas operativos.
PCB En electrónica, un circuito impreso o PCB (del inglés *Printed Circuit Board*), es un medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de *rutas* o *pistas* de material conductor, grabados desde hojas de cobre laminadas sobre un sustrato no conductor.
PLD (*Proggramable logic device)* Dispositivo de lógica programable
PROCEDURAL Estilo de programación de alto nivel con características de ejecución de instrucciones tipo serie.
PROM es el acrónimo de Programmable Read-Only Memory (ROM programable). Es una memoria digital donde el valor de cada bit depende del estado de un fusible.
PORTABLE Característica de un programa de computo para ser reutilizado.
PULL DOWN Termino empleado para denominar una resistencia limitadora de corriente que acopla hacía tierra física de un circuito.
RTL Register transfer level Grado de abstracción de circuitos a nivel registro.
SÍNTESIS Proceso de conversión de codigo HDL a instrucciones para implementar al diseño.
SMA Tipo de conector empleado para terminaciones de señales digitales, sobre todo de alta frecuencia.
SRAM. *Static random acces memory* Memoria de acceso aleatorio tipo estática
TESTBENCH Banco de pruebas digitales
TRENDING Grafica de comportamiento temporal de un sistema digital, tomando como referencia generalmente un ciclo de reloj.
TOOLBOX Paquete de herramientas de software
USB *Universal Serial Bus* Puerto Serie Universal
VGA El término Video Graphics Array (VGA) se refiere tanto a una pantalla de computadora analógica estándar; conector VGA de 15 clavijas D miniatura que se comercializó por primera vez en 1988 por IBM; o la resolución 640 × 480.
VHDL Es el acrónimo que representa la combinación de vhsic y hdl, donde VHSIC es el acrónimo de *Very High Speed Integrated Circuit* y HDL es a su vez el acrónimo de *Hardware Description Language*.
VHSIC (*very high speed integrated circuits)* Circuito integrado de alta velocidad
VLSI Acrónimo ingles de *Very Large Scale Integration*, integración en escala muy grande.
WARP2 Departamento de defensa de los Estados Unidos que fundó las bases del lenguaje VHDL.

FUENTES BIBLIOGRAFICAS

John F. Wakerly, *"Digital Design: Principles and Practices and Xilinx 4.2i Student Package (International Edition)", Third Edition,* USA, Prentice Hall, 2003, 650 pp.

R. de J. Romero-Troncoso, "Sistemas Digitales con VHDL**"** (México 2004).

Jorge Chávez, "Manual de Verilog"**, (**Marzo 1999).

Martin Hernández Hernández "Apuntes de Clase: Diseño de sistemas digitales" UNAM FES Aragón 2007.

Sitio de internet http://www.xilinx.com/literature**.** "Publicaciones referentes a la tarjeta Spartan 3E Starter Kit" 2007.

Sitio de internet http://www.asic-world.com. "Encoding State Machines".

Archivos de Ayuda del software: ISE Project Navigator & ACTIVE – HDL.